Sistema de Asistencia Interactiva en Lenguaje Natural

David Álvarez Martínez
Eduardo Cabezuelo Ortega
Alberto San Román Arteaga

Directora del proyecto
Belén Díaz-Agudo

Proyecto de Sistemas Informáticos
Curso 2005-2006

Facultad de Informática
Universidad Complutense de Madrid

Resumen

En este documento se describen los aspectos más relevantes del Sistema de Asistencia Interactiva que se ha diseñado. Este entorno presenta una interfaz textual que permite establecer comunicaciones con diferentes usuarios en lenguaje natural escrito. El objetivo principal de este trabajo consiste en hacer que el sistema "comprenda" los mensajes de los usuarios. Para llevar a cabo esta tarea, se utilizan técnicas de análisis y extracción de información, entre las que se encuentran el uso de ontologías del dominio y otros recursos lingüísticos como WordNet. Este sistema debe transformar las consultas textuales de los usuarios en una representación estructurada que contenga la información importante. Una vez finalizada esta tarea, el sistema debe buscar con estos datos una respuesta acorde que pueda ser devuelta al usuario que envió la consulta.

Para llevar a cabo la implementación de este sistema se ha utilizado el entorno jCOLIBRI, creado por el grupo de investigación GAIA de la propia Facultad de Informática. Este entorno provee una serie de métodos de procesamiento de lenguaje natural que han sido reutilizados para nuestro sistema, así como su principal utilidad consistente en la creación de sistemas con razonamiento basado en casos genéricos, usada como módulo de generación de respuesta, aunque es posible utilizar cualquier otro componente que realice la misma tarea.

El sistema se ha ejemplificado con el dominio de los viajes, aunque se puede aplicar a cualquier dominio específico.

Palabras clave

lenguaje natural, extracción de información, cbr, ontología, jcolibri, wordnet, procesamiento de textos

Abstract

This paper describes the most relevant aspects of the Interactive Assistance System designed. This environment provides a textual interface which allows it to communicate with users in written natural language. The main target of this project is to make the system capable of “understanding” user messages. To deal with this task, information extraction and analysis techniques are used; among them, domain ontologies and other linguistic resources such as WordNet. This system must transform user textual requests into structured representations containing the relevant information. Once this task is accomplished, the system must seek an appropriate answer to be sent to the user.

To implement this system, the jCOLIBRI environment, developed by the GAIA research group from this very Computer Science Faculty, has been used. This environment provides a series of natural language processing methods which have been reused in our system, as well as its main utility, which allows the creation of system with case-based reasoning, used as an answer generation module, although it is possible to use any other component to perform that same task.

The system has been exemplified in the travelling domain, although it can be applied to any other one.

Keywords

natural language, information extraction, cbr, ontology, jcolibri, wordnet, text processing

Los alumnos David Álvarez Martínez, Eduardo Cabezuelo Ortega y Alberto San Román Arteaga autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

En Madrid, a 6 de Julio de 2006

David Álvarez Martínez Eduardo Cabezuelo Ortega Alberto San Román Arteaga

Índice

Apartado 1
Introducción

Introducción

La idea esencial del trabajo a realizar en este proyecto es el desarrollo de un sistema que preste un servicio de atención a todos sus posibles clientes. Esta comunicación entre los usuarios y el sistema se debe realizar en lenguaje natural escrito, de forma que ambas partes se entiendan.

El **Procesamiento del Lenguaje Natural** (del inglés Natural Language Processing, NLP) es un campo de estudio y desarrollo de la Inteligencia Artificial y la Lingüística. Se ocupa de la formulación e investigación de mecanismos eficaces que permitan procesar de forma correcta la comunicación entre personas o entre personas y máquinas por medio de lenguajes naturales, independientemente de todas las vías de percepción posibles. El NLP cubre una gran cantidad de campos como el reconocimiento del habla, la traducción automática o la generación de lenguajes naturales, y entre los que se incluyen los dedicados al **Análisis y Compresión del Lenguaje** y la **Extracción de Información**, que serán en los que nosotros nos desenvolvamos a la hora de desarrollar nuestro trabajo en este proyecto.

La Extracción de Información (del inglés Information Extraction, IE) tiene como finalidad la extracción automática de información estructurada o semi-estructurada desde documentos sin estructura legibles por una máquina. La importancia de la IE está determinada por la creciente cantidad de información no estructurada, por ejemplo en Internet, existiendo hoy en día en todo el mundo más datos almacenados en forma de texto que en cualquier otra forma (como, por ejemplo, bases de datos relacionales o registros de transacciones bancarias). Este conocimiento podría hacerse más accesible extrayendo sus datos relevantes y almacenándolos en modelos estructurados.

La Extracción de Información desempeñará en nuestro proyecto un papel fundamental, ya que se trata de la técnica a aplicar sobre las consultas textuales de los usuarios que el sistema reciba, siendo nuestro principal objetivo la extracción de la información relevante de éstas para poder generar una respuesta con sentido.

El procesamiento del lenguaje natural siempre ha sido una de las piedras angulares de la Inteligencia Artificial, aunque su importancia dentro de este campo no ha sido siempre el mismo, como consecuencia de diversos cambios tecnológicos y de los resultados obtenidos en diferentes estudios y desarrollos. A pesar de todas las investigaciones y estudios realizados, hoy en día el procesamiento del lenguaje natural continúa siendo un problema no resuelto completamente, aunque sí se han conseguido una gran cantidad de avances significativos que nos permitirán desarrollar nuestro trabajo basándonos en ellos.

Siendo conscientes de las grandes dificultades que nos plantea el procesamiento del lenguaje natural, nuestra intención es desarrollar un sistema que se encargue de atender a los usuarios de un determinado

servicio como lo haría, por ejemplo, un departamento de atención al cliente de una empresa, al que se le pueda realizar todo tipo de consultas y éste las responda de forma coherente. Las ventajas de esta utilidad son claras, como por ejemplo la automatización de los sistemas de atención al cliente de las empresas o la posibilidad de prestación del servicio las 24 horas los 365 días del año. Aunque también hay que tener en cuenta las posibles limitaciones, en las que siempre estará implicado el tratamiento del lenguaje natural.

La idea de este sistema consiste el desarrollo de un entorno genérico que permita ser adaptado a cualquier dominio posible: reservas en hoteles, información sobre viajes, información sobre notas en facultades... De esta forma, el sistema podría ser adaptable a cualquier necesidad siempre que se le proveyera con el conocimiento del dominio específico sobre el que se quisiera trabajar.

Aunque la finalidad principal de nuestro trabajo es hacer que el sistema analice y extraiga información relevante de las consultas textuales de los usuarios y genera una respuesta adecuada, la comunicación entre ambos interlocutores podría realizarse a través de cualquier medio disponible como, por ejemplo, mediante el intercambio de correos electrónicos o con la utilización de mensajería instantánea. Así, el sistema debe permitir también en este aspecto total flexibilidad.

A lo largo de todo este documento se explicarán cada una de las etapas que se han llevado a cabo durante el desarrollo de nuestro proyecto. A continuación se describen brevemente cada uno de los apartados que componen esta memoria:

- **Requisitos y objetivos iniciales.** En este primer apartado se describen los primeros pasos tomados en el diseño y desarrollo de nuestro proyecto. En él, se realiza una pequeña introducción sobre el tratamiento del lenguaje natural de forma automática, para pasar después a explicar el objetivo fundamental del trabajo a realizar.

- **Conceptos teóricos.** En este apartado se introducen los conceptos teóricos fundamentales que es necesario conocer para entender el funcionamiento de todas las partes de nuestro sistema. En él se explican las bases del razonamiento basado en casos (CBR), los conceptos básicos de las ontologías y los fundamentos teóricos del entorno jCOLIBRI.

- **Fases de desarrollo.** En esta sección se explicarán cada una de las aproximaciones que se han desarrollado a lo largo de todo el proyecto. Estas son las partes fundamentales en que se ha dividido el trabajo, en cada una de las cuales se incluyen nuevas utilidades y funcionalidades al sistema:

    - **Diseño inicial del sistema.** En esta primera etapa del proyecto se realiza el primer diseño del sistema, que permitirá a sus usuarios introducir una consulta textual, que será analizada y de donde se extraerá la información relevante que permitirá generar una respuesta adecuada.

    - **Utilización de sinónimos.** En esta segunda aproximación se añade una nueva utilidad que permite buscar información relacionada en la consulta del usuario mediante el uso de sinónimos, a partir de la utilización de diccionarios y definiciones propias.

    - **Confirmación de información extraída.** En esta tercera etapa del proyecto el sistema permitirá al usuario confirmar la información extraída de su consulta, de forma que éste pueda corregir los datos, en caso de que considere que son erróneos.

    - **Utilización de ontologías.** En la cuarta aproximación se añade una gran funcionalidad al sistema que le permitirá, mediante la utilización de ontologías, obtener información relacionada con los datos extraídos de la consulta del usuario, de forma que sea posible generar respuestas alternativas si no se encuentran resultados en la búsqueda de casos.

- **Estudio de los resultados obtenidos.** En este apartado se realiza un estudio de los resultados obtenidos en cada una de las aproximaciones anteriores del proyecto, de forma que quede constancia las mejoras considerables que han tenido lugar en el sistema mediante la inclusión de nuevas funcionalidades en cada una de ellas.

- **Conclusiones.** En este último apartado se detallan las más importantes conclusiones obtenidas con el todo el trabajo realizado a lo largo de este proyecto.

Apartado 2
Requisitos y objetivos iniciales

Especificación de requisitos y objetivos iniciales

Según se ha explicado en la introducción, la funcionalidad principal de este proyecto es el diálogo de nuestro sistema con sus usuarios, a través del lenguaje natural escrito. Como ya sabemos, el lenguaje es el medio principal por el cual las personas se comunican, y que nos permite transmitir el conocimiento que poseemos del mundo.

En [01] se describen las principales dificultades en el estudio del lenguaje, que son, a su vez, las grandes ventajas que presenta este medio de comunicación:

- **Descripciones incompletas.** Las sentencias en el lenguaje natural son incompletas, ya que mucha información está implícita en el contexto.
- **Ambigüedad de significado.** Distintos significados de una misma expresión en diferentes contextos.
- **No completitud.** Es imposible disponer de un sistema de lenguaje natural completo, ya que continuamente se incorporan nuevas palabras, expresiones y significados.
- **Ambigüedad de expresión.** Generalmente, existen múltiples formas de expresar un mismo concepto.
- **Dependencia del idioma.** El procesamiento del lenguaje natural se realiza de formas distintas según qué idioma se esté utilizando.

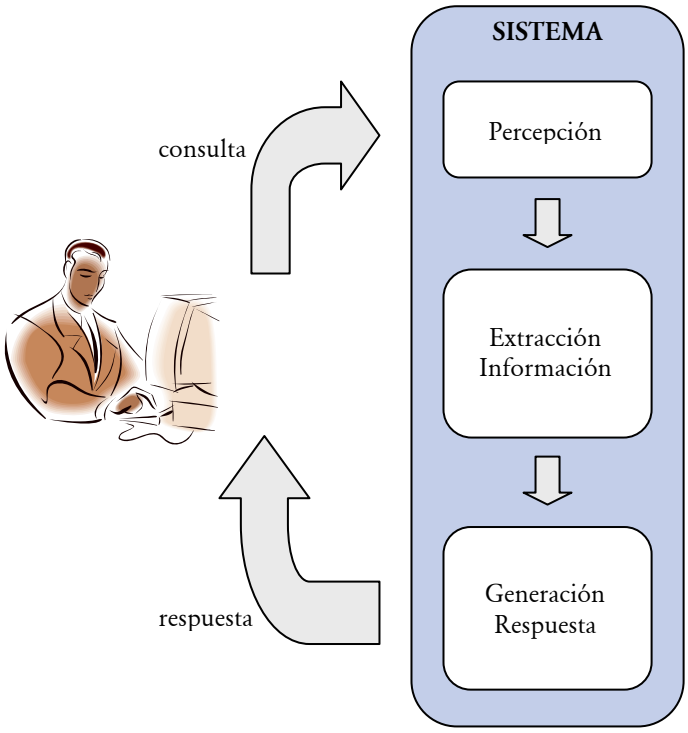
Considerando los puntos anteriores podemos darnos cuenta que el procesamiento del lenguaje natural no es un problema trivial con una solución sencilla. Debido a ésto, debemos ser conscientes de las serias limitaciones a las que nos enfrentamos y las dificultades que encontraremos a lo largo del desarrollo de nuestro proyecto.

Así mismo, en [01] se definen las siguientes fases en una comunicación:

- **Percepción.** A través de distintos tipos de vías: auditiva, visual o escrita.
- **Análisis.** Se obtiene el significado de palabras y frases, teniendo en cuentas que pueden tener varios significados (ambigüedad). Esta etapa se divide a su vez en distintas fases:
    - **Fase sintáctica.** En esta etapa se realiza un análisis morfológico y otro sintáctico para poder obtener la estructura de las frases.
    - **Fase semántica.** En esta etapa se interpretan las estructuras obtenidas en la fase anterior para poder otorgarlas un significado. Para realizar esta tarea es necesario tener en cuenta que el significado de una frase puede depender tanto de las frases anteriores como del contexto general (integración del discurso).
    - **Fase pragmática.** En esta etapa se resuelve e interpreta la fase para saber qué se debe hacer.
- **Eliminación de la ambigüedad.** Este es uno de los mayores problemas en el procesamiento del lenguaje, ya que es necesario encontrar la interpretación correcta dentro de todas las posibles.
- **Incorporación.** El agente receptor de la comunicación decide si incorporarla o no a su conocimiento.

Para nuestro sistema en concreto, la fase de "Percepción" se realiza a través de la vía escrita, ya que los usuarios escriben sus consultas, por tanto nos estamos refiriendo al tratamiento del lenguaje natural escrito. Las fases de "Análisis" y "Eliminación de la ambigüedad" componen la etapa de extracción de

información de nuestro sistema, en la que se obtendrán los datos relevantes a partir de las diferentes entradas que se reciban. En la definición del sistema no vamos a incluir la etapa de "Incorporación", ya que se dejará como una posible nueva aproximación para un futuro en la que se dote al sistema de una mayor "inteligencia" que le permita añadir a su conocimiento nueva información de forma dinámica.



Cada una de estas tres etapas realiza un trabajo específico:

- **Percepción.** En la primera etapa se obtienen las consultas de los usuarios del sistema. Como ya hemos comentado antes, la percepción se realiza a través de la vía escrita, es decir, las consultas son textuales. La representación de la información en este primer punto es "plana", ya que está contenida dentro de los textos de los usuarios.

- **Extracción de información.** En esta segunda etapa se aplican técnicas de análisis y extracción de información sobre las consultas recibidas por la percepción. En este caso, la representación de la información pasa de ser plana a estar estructurada en campos que almacenan los datos extraídos de los textos de los usuarios.

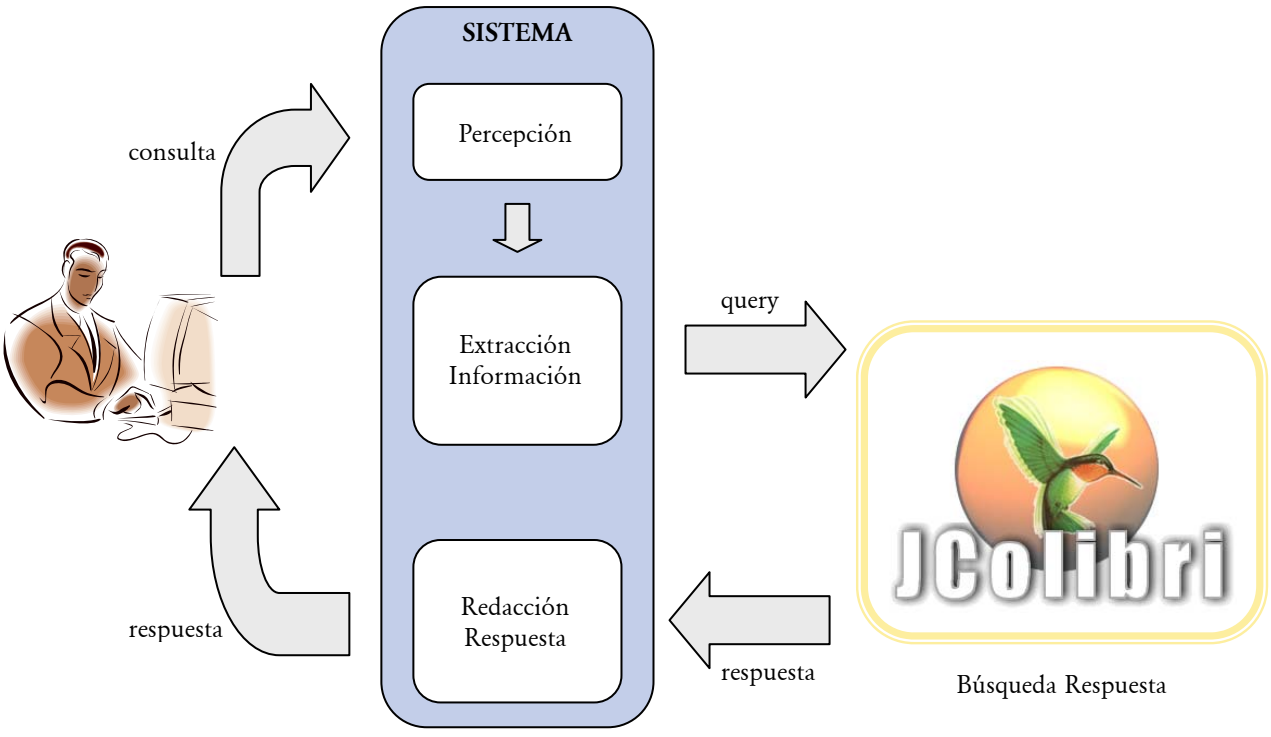
- **Generación de respuesta.** La última etapa del sistema recibe como entrada una estructura con los datos relevantes de la consultas de los usuarios y su objetivo es, a partir de éstos, obtener una respuesta adecuada. Para ello puede realizar múltiples acciones: búsquedas en bases de datos, razonamiento basado en casos...

Nuestro trabajo en este proyecto se va a centrar en la realización de los dos primeros módulos del sistema: "Percepción" y "Extracción de Información", con el comportamiento que se ha especificado en el párrafo anterior. La idea para el módulo de "Generación de Respuesta" es reutilizar un componente que se comporte de la forma que nosotros deseamos, es decir, que a partir de una estrada de datos estructurados sea capaz de encontrar una respuesta adecuada en el dominio específico en el que se esté trabajando. Según ya se ha explicado antes, para conseguir este resultado es posible realizar una gran cantidad de acciones, como consultas contra bases de datos. Sin embargo, existe una disciplina que nos

proporciona mejores resultados en la búsqueda de soluciones: el **Razonamiento Basado en Casos** (Case Based Reasoning, CBR).

"Un razonador basado en casos resuelve problemas nuevos mediante la adaptación de soluciones previas usadas para resolver problemas similares" ([03]). Este tipo de sistemas están formados por casos relativos al dominio en el que se trabaja. Su principal ventaja es que el CBR es un modo natural de razonamiento de los seres humanos. Como, además, nuestra pretensión es construir un sistema genérico y adaptable a todo tipo de dominios, el razonamiento basado en casos satisface totalmente nuestras necesidades, gracias a su representación del conocimiento en forma de casos totalmente adaptables a diferentes necesidades Por tanto, nuestra etapa de "Generación de Respuesta" será llevado a cabo por un sistema CBR que, dado una entrada de datos estructurada, encuentre una o varias soluciones adecuadas y las devuelva como resultado. Es importante destacar que las tareas que se realizan en este módulo podrían ser implementadas por cualquier otro tipo de disciplina (razonamiento basado en reglas, consultas sobre ontologías, sobre bases de datos. Nuestro sistema permite, por tanto, la adaptación de componentes para realizar sus tareas.

Para llevar a cabo el desarrollo de un sistema CBR encargado de la etapa de "Generación de Respuesta", desde el grupo de investigación **GAIA** (Group for Artificial Intelligence Applications, [11]) de la Facultad de Informática de la UCM se nos sugirió que utilizásemos un entorno orientado a objetos creado por ellos y destinado al desarrollo de sistemas CBR, llamado **jCOLIBRI** ([13]). Con él es posible crear este tipo de sistemas de forma genérica sobre cualquier dominio específico, precisamente lo que nosotros necesitamos para desarrollar el proyecto. Además, jCOLIBRI incluye una extensión para el procesamiento de textos, con métodos de análisis y extracción de información, aspecto que también es necesario en nuestro proyecto. Gracias a todas estas características, este entorno resulta perfecto como base para implementar todo el trabajo a realizar.

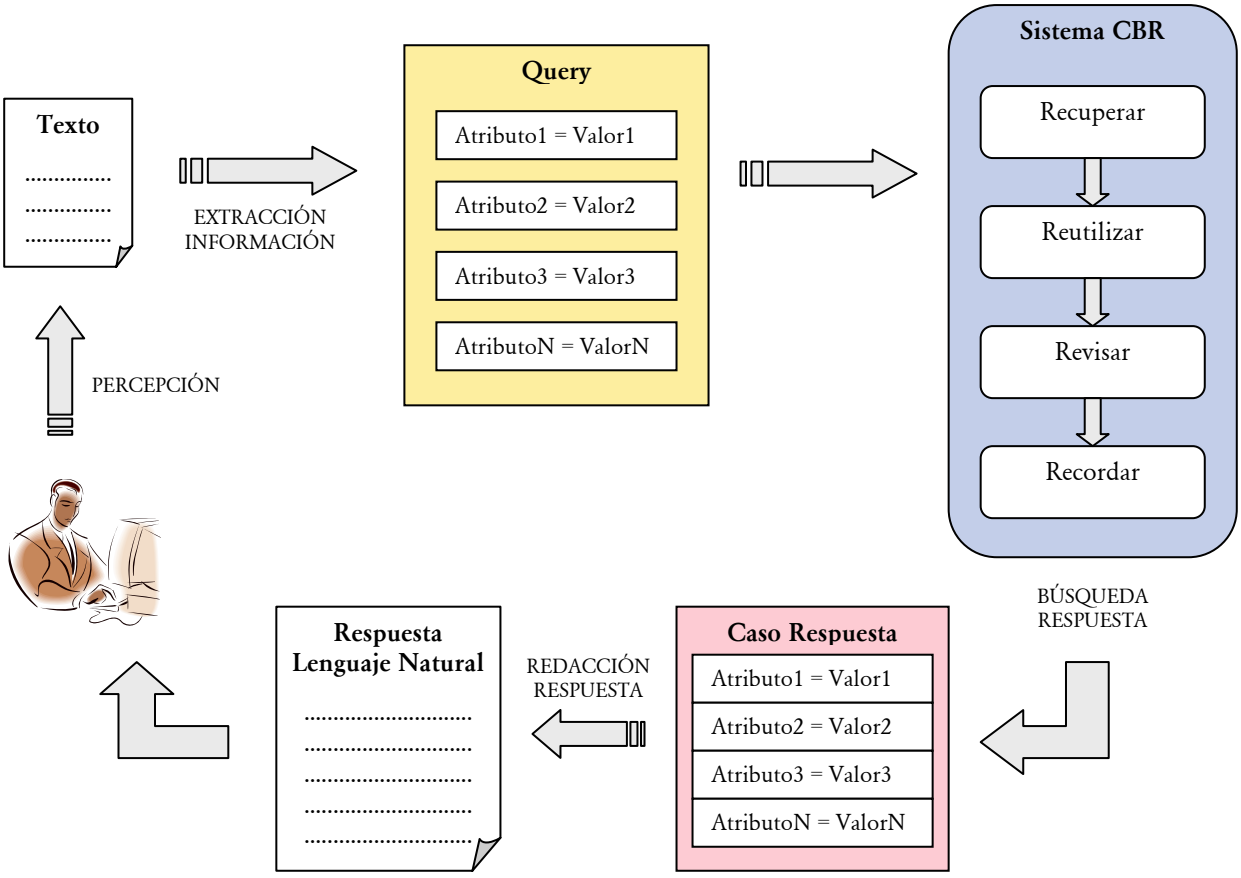


Como se puede ver en la figura anterior, el módulo de "Generación de Respuesta" de nuestro sistema ha sido sustituido por dos nuevos componentes: "Búsqueda de Respuesta" y "Redacción de Respuesta". El primero se encarga de, mediante la utilización del entorno jCOLIBRI y dados una serie de datos representados de forma estructurada (query), realizar una búsqueda en su base de conocimiento y devolver una respuesta adecuada. Esta solución, que será un caso del sistema CBR que tendrá la misma

estructura que la query recibida como entrada, llegará al segundo módulo, el de "Redacción de Respuesta", que se encarga de generar a partir de estos datos una respuesta en lenguaje natural que será retornada al usuario por el sistema como resultado del procesamiento de su consulta. Esta segunda etapa de redacción de una respuesta en lenguaje natural se deja como una alternativa de desarrollo de nuestro sistema, ya que su papel dentro de éste no es determinante. La respuesta devuelta por éste podría ser, perfectamente, la descripción de un caso mediante la enumeración de los valores que lo componen.

El entorno jCOLIBRI nos permite desarrollar sistemas con razonamiento basado en casos independientes de todo dominio, que lleven a cabo la etapa "Búsqueda de Respuesta" de nuestro sistema. Un sistema CBR consta de cuatro etapas, conocidas como las 4 R's: Recuperar, Reutilizar, Revisar y Recordar. El funcionamiento básico de este tipo de sistemas es el siguiente: dado un cierto caso de entrada se **recuperan** de la base de conocimiento aquellos casos con una mayor similitud con éste. Después, se **reutiliza** el conocimiento incluido en el caso recuperado para clasificar el problema actual y se **revisa** si la solución propuesta es correcta. Como último paso, el sistema **recuerda** el conocimiento adquirido. En el próximo apartado se extienden esto conceptos.

Como se puede observar, el comportamiento de un sistema CBR coincide con el esperado en nuestra etapa de "Búsqueda de Respuesta" que hemos descrito en párrafos anteriores, es decir, el CBR nos permite encontrar la respuesta más adecuada dada una cierta entrada entre todos los casos que forman parte de la base de conocimiento del sistema. Para poder realizar esta acción es necesario que la entrada también sea un caso con la misma estructura que todos los de la base de casos, de forma que se pueda aplicar la etapa de "Recuperación" mediante el uso de funciones de similitud. Por ello, previo al módulo CBR es necesario añadir una etapa previa de transformación de la entrada de un texto de consulta de un usuario a un caso con la misma información pero almacenada de forma estructurada.



Como se puede observar en la figura anterior, el usuario utiliza la percepción del sistema para escribir su consulta en forma de texto. Una vez éste está completo, el sistema extrae la información relevante y

completa una estructura compuesta por una serie de atributos con sus correspondientes valores, denominada query, siempre que éstos hayan sido encontrados en el texto del usuario. Al realizar esta transformación en los datos de entrada ya pueden ser utilizados por el sistema CBR para buscar en su base de conocimiento el caso que guarde una mayor similitud con la query que recibe, para devolverlo como respuesta, con la posibilidad de redactar previamente un texto en lenguaje natural. Es importante recordar en este punto que el módulo CBR, proporcionado por jCOLIBRI, puede ser sustituido por cualquier otro que sea capaz de encontrar una respuesta adecuada dada una cierta entrada procedente del usuario. Nuestro trabajo se centra, por tanto, en la implementación de una **interfaz textual** que transforme un texto plano en una estructura con los datos relevantes de ésta. Esta nueva representación de la información de la consulta del usuario puede ser tomada por cualquier módulo que genere a partir de ella una respuesta.

En los próximos apartados de este documento se explican en detalle todos los aspectos de nuestro sistema que han sido introducidos en esta sección, desde su implementación hasta las pruebas que han sido realizadas para comprobar su funcionamiento óptimo.

A continuación se detallan el resto de requisitos y objetivos iniciales de nuestro proyecto, así como el un pequeño estudio con trabajos relacionados.

Dominio de trabajo

Aunque nuestro sistema se va a diseñar de forma genérica para cualquier dominio específico, es necesario definir uno que nos permita trabajar sobre él y desarrollar cada una de las aproximaciones del proyecto, así como realizar diferentes pruebas para comprobar su óptimo comportamiento.

El entono jCOLIBRI proporciona varios dominios de ejemplo con extensas bases de casos. Algunos de estos dominios son sobre restaurantes (casos en forma de texto sobre los que es necesario extraer la información con que rellenar los valores de los atributos) y sobre **viajes** (con la información ya extraída, es decir, estructurada). Éste último fue el seleccionado por nosotros.

En este caso, la información que conforma la base de casos del sistema es almacenada en una base de datos. Esta estructura proporcionada 1024 viajes con una serie de atributos comunes en todos. Estos atributos toman diferentes valores para cada uno de los casos. A continuación se detallan estos campos en los que se divide cada viaje:

- **Identificador del viaje.** Sirve para identificar de forma única a cada uno de los casos.
- **Tipo de vacaciones.** Indica el tipo de actividades realizadas en el viaje. De tipo enumerado.
- **Precio.** Indica el precio del viaje. De tipo numérico.
- **Número de personas.** Indica el número de personas que admite el viaje. De tipo numérico.
- **Región.** Indica el destino del viaje. De tipo enumerado.
- **Transporte.** Indica el tipo de transporte utilizados durante el viaje. De tipo enumerado.
- **Duración.** Indica la duración del viaje en días. De tipo numérico.
- **Época.** Indica el mes o meses del año en que se realiza el viaje. De tipo enumerado.
- **Alojamiento.** Indica la categoría del alojamiento del viaje. De tipo enumerado.
- **Hotel.** Indica el nombre del hotel donde se alojan a los viajeros. De tipo cadena de texto.

Cada uno de los 1024 casos de la base de casos contiene todos los atributos anteriores, aunque para cada uno de ellos toman valores diferentes. La finalidad de nuestro sistema es comparar las similitudes que existen entre la consulta del usuario y cada uno de estos casos y devolver el viaje más acorde con los gustos y preferencias de éste.

Es importante destacar que la base de casos de viajes proporcionada por el entorno jCOLIBRI está escrita en **inglés**, por lo que las consultas de los usuarios al sistema tendrán que ser realizar en el mismo

idioma. Ésto no restringe a nuestro sistema en cuanto a su flexibilidad, ya que será totalmente adaptable a cualquier idioma o dominio específico.

Conceptos teóricos

Para llevar a cabo el desarrollo de nuestro sistema es necesario el aprendizaje de determinados conceptos teóricos imprescindibles. A continuación se detallan algunos de los más importantes:

- En primer lugar es necesario realizar un estudio en profundidad del entorno **jCOLIBRI**. Este aprendizaje se puede realizar a través de los tutoriales y vídeos proporcionados con ésta ([12]). En este apartado también se incluye el estudio de la extensión textual del entorno.

- Estudio de los fundamentos más importantes sobre el **razonamiento basado en casos**. Una buena referencia son los apuntes ([14]) de la asignatura optativa de nuestra titulación Ingeniería de Sistemas Basados en Conocimiento (ISBC), donde se describen los conceptos fundamentales, aunque siempre se puede ampliar con más información de libros y páginas web.

- Estudio de los conceptos principales de las **ontologías**, necesarias para la inclusión de una mayor cantidad de conocimiento en nuestro sistema. La mejor fuente para conocer los conceptos básicos es el completo tutorial ([08]) desarrollado por la Universidad de Stanford.

Posibles arquitecturas

Por último, en este apartado se estudian diversas arquitecturas de diseño y la posición de nuestro proyecto dentro de éstas. Las principales opciones consideradas son las siguientes:

- **Arquitectura basada en agentes inteligentes.** Este tipo de arquitectura está compuesta por una serie de componentes autónomos, denominados agentes, cada uno de los cuales tiene una tarea específica. Además, se incluyen una serie de recursos que realizan tareas como interacción con una base de datos (almacenamiento y extracción de información) o gestión de la visualización de la aplicación. Generalmente, estas arquitecturas se diseñan con una jerarquía de agentes, con uno encargado de la organización, por encima de otros dos que controlan el resto de agentes y recursos del sistema.
  Para llevar a cabo el desarrollo de esta aproximación se pensó en la utilización del entorno JADE (Java Agent DEvelopment Framework, [17]), que proporciona un protocolo estándar de diseño y comunicación entre agentes inteligentes. Una segunda opción sería utilizar una biblioteca de componentes (agentes y recursos) de la que disponemos.

- **Arquitectura orientada a servicios.** Este tipo de arquitecturas definen la utilización de servicios para dar soporte a los requerimientos de software del usuario. En un ambiente SOA, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado.
  Para esta aproximación se han barajado diferentes alternativas, como Servicios Web (Web Services, [29]), BPEL (Business Process Execution Language, [32]) o JSP (Java Server Pages, [31]).

Para este tipo de proyecto, una arquitectura orientada a servicios nos parece mucho más ventajosa, ya que es posible desplegar un servicio que reciba vía web consultas de usuarios de cualquier parte del mundo, las analice, extraiga la información pertinente y envíe de vuelta los resultados obtenidos. Aún así, la elección de la arquitectura más propicia se dejará para la última parte del proyecto.

En los apartados siguientes de este documento se explican los conceptos teóricos principales en los que se basa nuestro sistema. Estos conceptos son el desarrollo de ontologías y el razonamiento basado en casos. También se realiza una pequeña introducción del entorno jCOLIBRI.

En el siguiente sub-apartado se describen algunos de los trabajos relacionados con nuestro proyecto, lo que es conocido como el "Estado del arte".

Trabajos relacionados

Son muchos los trabajos e investigaciones que se han llevado a cabo a lo largo de los últimos años en el extenso campo del Procesamiento del Lenguaje Natural. Sin embargo, nosotros nos vamos a centrar en aquellos con una mayor similitud con nuestro proyecto, cuya funcionalidad principal sea la Extracción de Información, correspondiente a nuestro trabajo de implementación de una interfaz textual para sistemas CBR. Aunque, como ya hemos dicho, son muchos los proyectos existentes, sólo se van a listar una parte representativa.

Las referencias utilizadas para la documentación de estos trabajos se encuentran en la parte final de esta memoria, en el apartado de Bibliografía.

Algunos de los trabajos, ordenados alfabéticamente, relacionados con nuestro proyecto son:

- ACE ([21]).
- ANNIE ([27]).
- Ask Jeeves ([19]).
- FASTUS ([22]).
- GENIA ([23]).
- Pinocchio ([25]).
- START ([19]).
- VisualText ([24]).

ACE

El objetivo de este programa es el desarrollo de tecnología de extracción automática de contenido que permita el procesamiento del lenguaje humano en forma de texto. Se centra en tres tipos de fuentes: teletipos, emisión de noticias y periódicos. Su tecnología R&D tiene la finalidad admitir distintos tipos de aplicaciones de clasificación, filtrado y selección mediante la extracción y representación contenido del lenguaje.

ANNIE

Se trata de un proyecto de extracción de información portable con la finalidad de que pueda ser utilizado en muchas aplicaciones diferentes, tipos de texto o propósitos. Esta potabilidad implica una serie de condiciones, como poder tratar textos en múltiples formatos (correos electrónicos con faltas ortográficas, documentos estructurados HTML o XML...), procesar grandes cantidades de información a una gran velocidad o en cualquier idioma.

ANNIE es utilizado en el análisis de comentarios, artículos o páginas web sobre fútbol con la finalidad de clasificar vídeos de partidos. También se usa para crear sumarios de empresas en cuestiones de salud y seguridad.

Ask Jeeves

La idea original de este conocido buscador consistía en la capacidad de contestar a preguntas hechas mediante procesamiento de lenguaje natural. Ask Jeeves fue el primer motor de búsqueda comercial capaz de contestar preguntas en Internet. Su finalidad es dar respuesta a una gran variedad de preguntas en inglés llano, esforzándose en ser una aplicación sencilla e intuitiva.

FASTUS

Se trata de un sistema de extracción de información sobre texto libre, actualmente para los idiomas inglés y japonés. Este proyecto fue diseñado en respuesta a las necesidades de procesamiento de grandes cantidades de texto escrito sobre cualquier dominio específico. Ha estado en desarrollo desde 1992 y está implementado en Lisp.

GENIA

Este proyecto tiene como objetivo la extracción automática de información relevante sobre textos escritos por científicos, orientados al campo de la microbiología, para ayudar a superar las limitaciones impuestas por las grandes cantidades de datos, muchas veces, inaccesibles. En la actualidad trabajan en un modelo de extracción de información sobre interacciones de proteínas.

Pinocchio

Se trata de un conjunto de herramientas destinadas al desarrollo y ejecución de aplicaciones de extracción de información. Pinocchio provee un entorno con una interfaz gráfica con editores especializados, depuradores y recursos de trazas. Además, es independiente del idioma ya que, aunque ha sido utilizado en aplicaciones para italiano, existen demostradores para inglés y ruso (de forma parcial).

Pinocchio permite el desarrollo de aplicaciones típicas de extracción de información sobre textos con el fin de rellenar plantillas con los datos relevantes.

START

Se trata del primer sistema que se diseñó capaz de responder a preguntas basado en Web. Se basa en el procesamiento del lenguaje natural y de la obtención de información sobre sistemas de recuperación y organización de datos, como motores de búsqueda. Actualmente, el sistema puede contestar millones de preguntas en inglés acerca de lugares (ciudades, países, lagos, mapas...), películas, personas, definiciones de diccionario y mucho más.

VisualText

Herramienta que permite un desarrollo rápido y preciso de sistemas de extracción de información, procesamiento de lenguaje natural y análisis de textos. Una de sus mayores utilidades consiste en rellenar bases de datos con información escondida en complejos textos.

VisualText puede ser utilizado, por ejemplo para extraer noticias de negocios de sitios web y crear una base de datos con la última información financiera.

También podemos destacar algunos sistemas representativos que utilizan el razonamiento basado en casos para la interpretación de soluciones o la resolución de problemas, al igual que el nuestro:

- CASEY ([28]).
- CHEF ([28]).
- HYPO ([28]).
- JULIA ([28]).

CASEY

Se trata de un sistema de diagnóstico médico basado en casos. Como entrada toma una descripción de un determinado paciente, incluyendo muestras normales y presentándolas como síntomas. Su salida es una explicación causal de los desórdenes que tiene el paciente. CASEY diagnostica los pacientes aplicando heurísticas de adaptación y emparejamiento basado en el modelo independientes del dominio.

CHEF

Se trata de un planificador basado en casos cuyo dominio se centra en la creación de recetas de cocina. Las recetas, que son vistas como planes, proporcionan la secuencia de pasos a seguir para preparar un plato. La entrada de este sistema son las metas alcanzables por las recetas (por ejemplo, incluye carne, sofreír, sabor dulce...) y la salida es una receta (plan), que puede obtener esas metas. Como planificador basado en casos, CHEF crea sus planes a partir de otros anteriores que funcionaron en situaciones similares, modificándolos con la finalidad de adaptarlos a la nueva situación.

HYPO

Se trata del primer modelo CBR en el campo legal. Es un sistema que analiza casos y construye argumentos legales para sus clientes. HYPO puede crear argumentos igual de buenos tanto para la defensa como para la acusación. El dominio particular en el que tiene más experiencia es el de los casos de revelación (un secreto de una compañía se desvela por otra de la competencia y ésta saca provecho de la situación). Para cada tema, HYPO indica qué lado de la disputa saldrá favorecido.

JULIA

Se trata de un sistema basado en casos que trabaja en el dominio de la planificación de comidas. Su principal funcionalidad consiste en determinar qué comida debe ser preparada en una determinada especificación. El número y el tipo de invitados, sus características o el espacio que se necesita son algunos de los factores que afectan a la elección de un menú. JULIA utiliza una combinación de restricciones y un razonamiento basado en casos para ayudar al proceso de diseño del menú.

Apartado 3
Conceptos teóricos

Razonamiento Basado en Casos (CBR)

El Razonamiento Basado en Casos, del inglés Case-Based Reasoning (CBR), en su definición más amplia, es el proceso de resolución de nuevos problemas mediante la utilización de soluciones de otros similares ya pasados. Por ejemplo, un abogado que integra en sus argumentaciones el veredicto de casos previos está utilizando razonamiento basado en casos. También los médicos que buscan conjuntos de síntomas que identifican al paciente con algún conjunto de casos previos o los ingenieros que toman muchas de sus ideas de soluciones previas ya construidas con éxito. El razonamiento basado en casos es un destacado tipo de resolución de problemas por analogía.

"Un razonador basado en casos resuelve nuevos problemas adaptando las soluciones que fueron utilizadas para resolver problemas previos similares" [03].

El razonamiento basado en casos puede ser considerados, además de como un método muy potente de razonamiento artificial, como un comportamiento dominante en la resolución de problemas en el día a día de las personas, basado en las analogías existentes entre situaciones similares.

Existen dos tipos de básicos de sistemas CBR:

- **Interpretación de situaciones.** Su objetivo es formular un juicio o clasificar una situación. Este tipo de sistemas es usado, por ejemplo, para aplicaciones al sistema judicial anglosajón o de diagnóstico de enfermedades, fallos en sistemas, etc.

- **Resolución de problemas.** Su objetivo es aplicar la solución a un problema pasado para obtener la solución al problema actual. Se utilizan, por ejemplo, para diseño y planificación.

Cada una de estas situaciones previas o soluciones (conocimiento) son conocidas en CBR como **casos**. "Un caso es un fragmento contextualizado de conocimiento que representa una experiencia y que enseña una lección importante para conseguir los objetivos del razonador"[Kolodner & Leake 97]. Es necesario tener en cuenta que no todos los casos aportan conocimiento útil, ya que pueden existir algunos que sean redundantes, con información repetida, y también otros que estén cubiertos por el conocimiento general del sistema.

Un caso está compuesto por la siguiente información:

- **Descripción de la situación o del problema.** En esta parte se definen objetivos, restricciones para la consecución de éstos y características de la situación.

- **Solución.** Aquí se incluye la descripción de la solución, los métodos que se siguieron para poder obtenerla y las justificaciones de las decisiones tomadas para llegar a ella.

- **Resultado.** En este punto se especifica si la solución aplicada tuvo éxito o no. Además, en caso de que fallara se debe incluir el motivo por el que fue.

Los casos que forman parte de la base de conocimiento de un sistema con razonamiento basado en casos pueden ser representados de múltiples formas. Éstas son las principales:

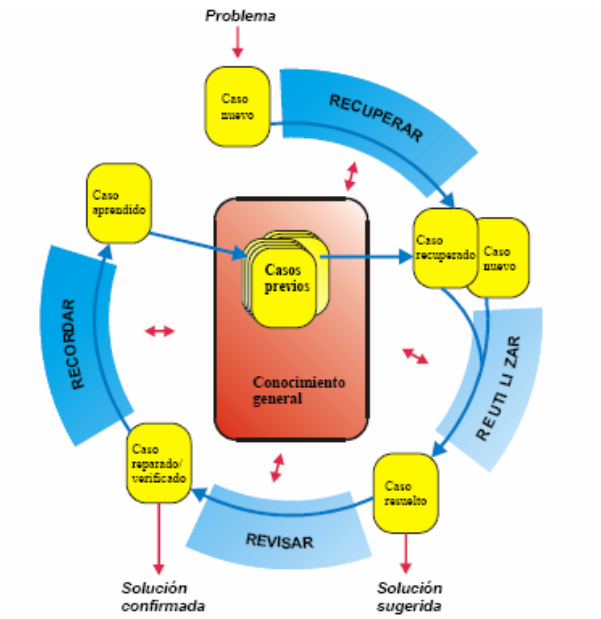
- Listas de pares atributo-valor.
- Registros de una base de datos.
- Sistemas de marcos y redes semánticas o modelos de memoria.
- Textos estructurados.

Además de la base de casos, formada por un conjunto de soluciones pasadas, un sistema CBR típico posee el siguiente conocimiento general:

- Conocimiento de similitud, utilizado para recuperar casos relevantes dado un cierto problema.
- Conocimiento de adaptación, normalmente en forma de reglas de sustitución.
- Modelos del dominio (ontologías), utilizados en las distintas fases del CBR para el cómputo de similitud y en búsqueda de sustitutos.

Ciclo CBR

El razonamiento basado en casos ha sido formalizado como un proceso compuesto por cuatro etapas, conocidas como las 4 R's. En la siguiente figura se muestra el esquema de un ciclo CBR.



A continuación se explican en detalle cada una de las cuatro etapas anteriores correspondientes a un ciclo de razonamiento basado en casos:

- **Recuperar (Retrieve).** Da un problema objetivo, se recuperan de la base de conocimiento del sistema una serie de casos que son relevantes para su resolución. Para llevar a cabo esta tarea, es necesario valorar la situación y determinar las características que permiten encontrar dichos casos relevantes.
  El procedimiento de búsqueda dependerá de la organización (estructuras de datos: árboles, listas, redes...) de los casos, aunque, en ocasiones, éstos se organizan de forma automática utilizando métodos de aprendizaje máquina.
  En este punto es posible realizar una ordenación de los casos recuperados para realizar una comparación más elaborada o, incluso, seleccionar el caso mejor.

- **Reutilizar (Reuse).** En esta etapa se utiliza el conocimiento incluido en el caso recuperado para resolver el problema actual. Puede que en este punto sea necesario adaptar su solución para que se ajuste a la nueva situación (resolución de problemas) o cambiar su justificación (interpretación de soluciones). Si no es necesario modificarla es que la solución es válida o es el usuario quien se encarga de adaptarla o interpretarla.

- **Revisar (Revise).** Una vez ajustada la solución previa al objetivo actual es necesario comprobar si esta propuesta es correcta. La evaluación de la solución se realiza, normalmente, fuera del sistema CBR, mediante la respuesta de un experto (o del usuario) o mediante técnicas de simulación.

- **Recordar (Retain).** Después de que la solución haya sido adaptada correctamente al problema objetivo es necesario almacenar la nueva experiencia resultante como un nuevo caso en la base de conocimiento del sistema. En el CBR el razonamiento y el aprendizaje están íntimamente ligados, ya que un sistema con este tipo de razonamiento mejora en el uso al ir adquiriendo nuevas experiencias que integra adecuadamente. Además, es posible mejorar la eficiencia del sistema al disponer de más casos a partir de los cuales obtener soluciones.

Para comprender mejor el comportamiento de un ciclo CBR consideremos un ejemplo en el mundo real. Supongamos que Julia desea preparar tortitas con arándanos. Como es una cocinera inexperta, la experiencia más parecida que puede recordar fue cuando preparó tortitas solas. El procedimiento que siguió para hacerlas, junto con las justificaciones de las decisiones que tomó, constituyen el caso recuperado de Julia (Etapa Recuperar).

Para poder preparar las tortitas con arándanos, Julia ha de adaptar la solución recuperada para incluir los arándanos (Etapa Reutilizar). Para ello, añade los arándanos a la masa. Después de mezclarlo todo, se da cuenta de que ésta se ha vuelto de color azul, un efecto no deseado). Por esta razón, Julia sugiere la siguiente revisión de la solución adoptada: retrasar la adición de los arándanos hasta el momento de freír las tortitas en la sartén (Etapa Revisar).

Por último, a Julia sólo le queda recordar el nuevo procedimiento aprendido para cocinar tortitas con arándanos, enriqueciendo así su conocimiento y sus experiencias (Etapa Recordar).

Principales conceptos sobre ontologías

Una **ontología** acepta una gran cantidad de definiciones. Una descripción formal de este tipo de estructuras es la incluida en el documento de la Universidad de Stanford "Ontology Development 101: A Guide to Creating Your First Ontology" ([08]):

"Una ontología es una descripción formal explícita de conceptos en un dominio del discurso (clases o conceptos), propiedades de estos conceptos, de los que describen características o atributos (propiedades, papeles o "slots") y restricciones para esas propiedades ("facets")".

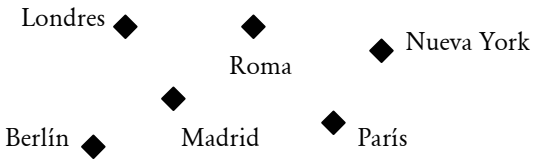
Las ontologías son utilizadas generalmente para representar el conocimiento de un cierto dominio de interés. Una ontología describe los conceptos de este dominio y también las relaciones existentes entre éstos.

Una ontología está formada por una serie de componentes: **Ejemplares** (también llamados Instancias o Individuales), **Propiedades** y **Clases** (también llamadas Conceptos).

Ejemplares

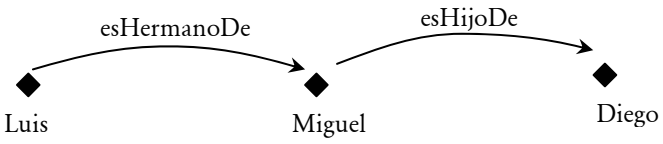
Los ejemplares en una ontología representan objetos del dominio en los que estamos interesados. Otra forma de llamar a un ejemplar es instancia (ya que pueden ser considerados como instancias de clases) o individual (de la traducción directa de "Individual" en inglés).

En la siguiente figura se representan una serie de individuales en un cierto dominio. Como convenio, en este documento se representarán los ejemplares con rombos.



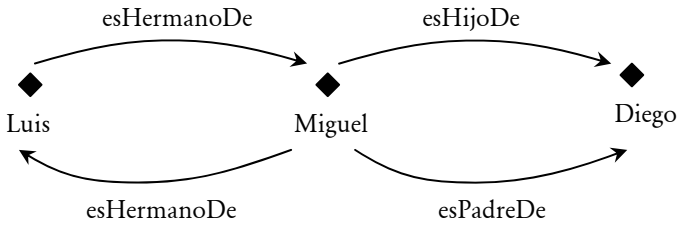
Propiedades

Una propiedad es una relación binaria (entre dos elementos) sobre ejemplares. Por ejemplo, la propiedad "esHermanoDe" puede unir el ejemplar "Luis" con el ejemplar "Miguel", o la propiedad "esHijoDe" puede unir el individual "Miguel" con "Diego".



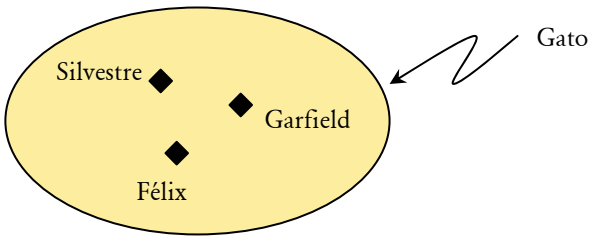
Es importante destacar que toda propiedad "va en una dirección", por lo que su significado no es el mismo si ésta se cambia. En el ejemplo, no es lo mismo decir que Miguel es hijo de Diego, que Diego es hijo de Miguel. En este caso se supone que Diego es el padre y Miguel el hijo, por lo que es necesario denotarlo así.

Una propiedad puede tener inversa. Por ejemplo, la inversa de la propiedad "esHijoDe" es "esPadreDe". En el caso de la propiedad "esHermanoDe", su inversa es ella misma (si Luis es hermano de Miguel, Miguel es hermano de Luis).



Clases

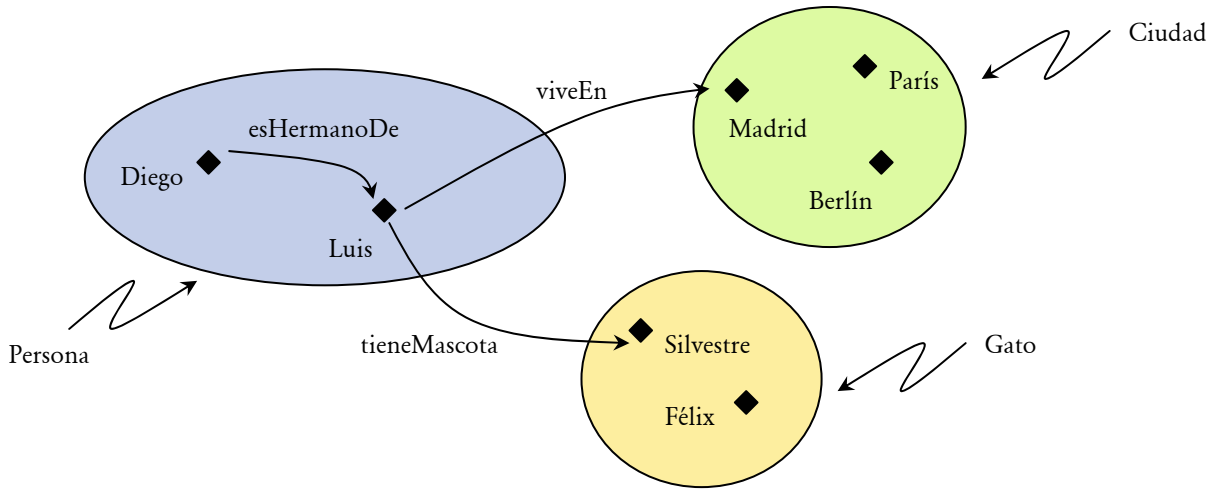
Las clases pueden ser consideradas como conjuntos que contienen instancias. Una clase es definida utilizando descripciones formales que especifican los requisitos necesarios para poder pertenecer a ellas. Por ejemplo, la clase "Gato" contendría todos los ejemplares de nuestro dominio que fueran gatos.



Las clases pueden ser clasificadas en una jerarquía de subclases y superclases, lo que recibe el nombre de **taxonomía**. Por ejemplo, si consideramos las clases "Animal" y "Gato", ésta última puede ser una subclase de la primera, lo que significa que "Animal" es una superclase de "Gato". Ésto significa que "Todos los gatos son animales" o que "Todos los miembros de la clase Gato son miembros de la clase Animal", por lo que "si eres un gato, eres un animal".

Muchas veces se utiliza la palabra "concepto" en sustitución de "clase", ya que se puede considerar que una clase es una representación concreta de conceptos.

En la siguiente figura se muestra una representación de algunas clases que contienen diversos ejemplares, unidos entre si mediante relaciones (o propiedades):



Fundamentos teóricos de jCOLIBRI

En este apartado se introducen los conceptos fundamentales del entorno de desarrollo jCOLIBRI, que será utilizado por nuestro sistema para la creación de sistemas con razonamiento basado en casos (CBR) que nos permitan encontrar respuestas a las consultas textuales de los usuarios.

jCOLIBRI (Java Cases and Ontology Libraries Integration for Building Reasoning Infrastructures) es un entorno orientado a objetos destinado al desarrollo de aplicaciones de razonamiento basado en casos (CBR). Un entorno, o framework, es una aplicación reusable y semi-completa que puede ser especializada para producir otras a medida.

jCOLIBRI está construido en torno a una ontología de métodos/tareas. Se trata de una descripción a nivel de conocimiento que guía el diseño del entorno, determina sus posibles extensiones y sostiene el proceso de creación de instancias. Estos métodos y tareas están descritos mediante una terminología CBR independiente del dominio, con una correspondencia con las clases del entorno.

En este entorno, una tarea puede estar descompuesta en una serie de sub-tareas más específicas. Existen métodos que permiten resolver tareas mediante la división de éstas en sub-tareas o mediante resolución directa. Para cada tarea existe un conjunto de métodos que la resuelven, y cada uno de estos métodos, a su vez, puede dividir su trabajo en otras sub-tareas. Detrás de esta estructura de tareas, el entorno jCOLIBRI incluye una librería de métodos de resolución de problemas (Problem Solving Method, PSM) para poder resolverlas.

El entorno jCOLIBRI es un artefacto que se basa en la idea de promover la reutilización de software para la construcción de sistemas con razonamiento basado en casos (sistemas CBR). El objetivo principal es integrar en la aplicación probadas técnicas de Ingeniería del Software con la idea básica de KADS de separar el método de resolución de problemas, que define el proceso de razonamiento, del modelo de dominio, que describe el conocimiento específico del dominio. jCOLIBRI es una evolución de la arquitectura de COLIBRI, cuya estructura consistía en una librería de PSMs, que permitía la resolución de tareas de un sistema CBR intensivo en conocimiento, conjuntamente con una ontología, CBROnto, con terminología CBR común.

Uno de los grandes problemas que existe con los entornos de desarrollo es el aprender a usarlos. Es por esto que jCOLIBRI proporciona una herramienta semi-automática de configuración que va guiando el proceso de instanciación a través de un interfaz gráfico. La configuración de un sistema CBR que usa este interfaz consta de los siguientes procesos:

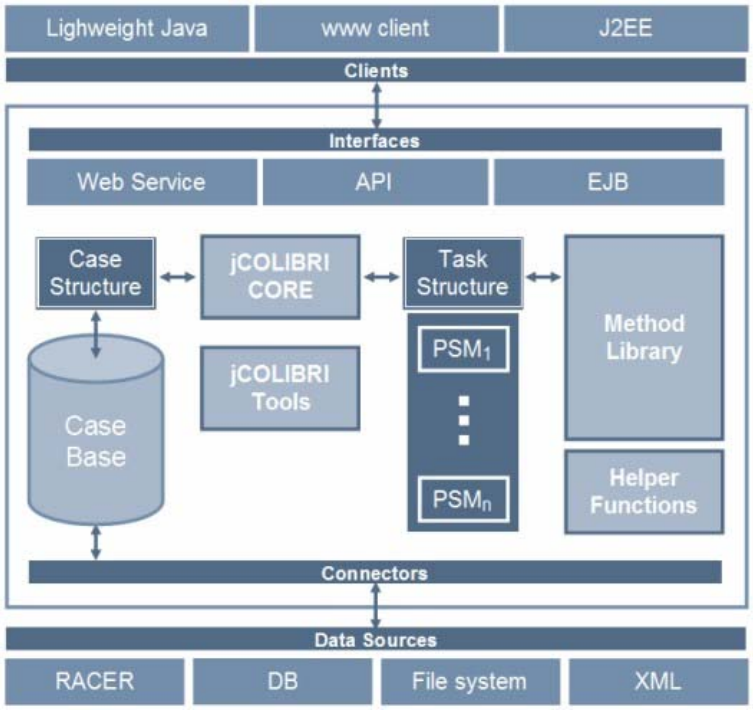
- Definición de la estructura de casos, la fuente de información que permite crear los casos y la organización de las base de conocimiento.

- Mientras el sistema no esté completo, se escoge una de las tareas sin un método asignado y se selecciona y configura un método para esa tarea. Al comenzar, el árbol de tareas posee sólo un elemento, CBRTask, que se resuelve mediante un método de descomposición del que resultan tareas adicionales. Las restricciones de las tareas/métodos se van siguiendo durante el proceso de configuración de manera que solamente los métodos aplicables en el contexto dado estarán disponibles para el diseñador del sistema.

- Una vez que el sistema está configurado, se genera el código de configuración que hace esté disponible el sistema CBR. La herramienta de configuración también proporciona un interfaz por defecto para hacer funcionar el sistema CBR, aunque en una situación real debería desarrollarse un interfaz específico para la aplicación.

Arquitectura de jCOLIBRI

La arquitectura del entorno jCOLIBRI comprende una jerarquía de clases Java más un cierto número de ficheros XML que permiten configurar las aplicaciones CBR generadas. La siguiente figura ilustra la arquitectura de jCOLIBRI.



Tareas y Métodos

El análisis a nivel de conocimiento que se aplica a los sistemas CBR describe el ciclo general CBR en términos de cuatro tareas al mayor nivel de generalidad:

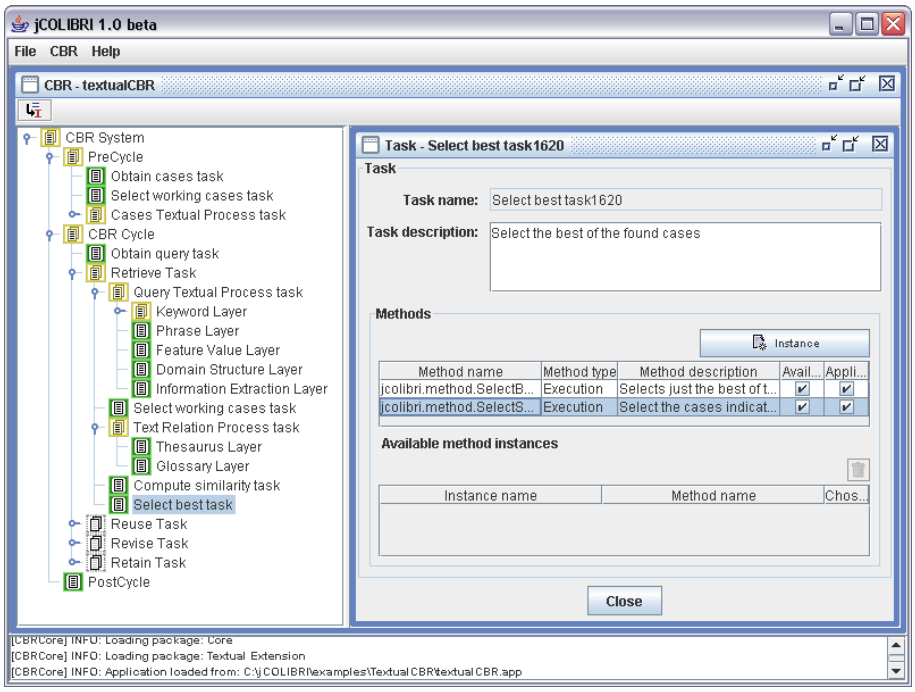
- Recuperar los casos más similares.
- Reutilizar ese conocimiento para solucionar el problema.
- Revisar la solución propuesta.
- Recordar esa experiencia.

Cada una de estas cuatro etapas de un ciclo CBR conlleva la utilización de un determinado número de sub-tareas específicas. Existen métodos para la resolución de tareas mediante la descomposición de estas en sub-tareas (métodos de descomposición) o resolviéndolas directamente (métodos de resolución).

Desde el punto de vista práctico, las tareas describen la funcionalidad que será resulta por el programador del CBR, definiendo qué código debe ser ejecutado. Los métodos resuelven estas tareas y ejecutan el código CBR. Como estos métodos deben ser reutilizables no pueden utilizar información dependiente del dominio. De esta manera, son implementados como Métodos de Resolución de Problemas (PSM's): métodos independientes del dominio que pueden ser configurados con la información del un determinado dominio específico para la creación de una aplicación completa.

La siguiente figura representa la estructura de descomposición en tareas que se realiza en el entorno. La mayoría de los sistemas CBR necesitan llevar a cabo determinadas tareas con el fin de preparar el ciclo de ejecución principal (ciclo CBR). Estas acciones de preparación se encuentran al mismo nivel que las

tareas principales del CBR (las 4 R's) y son ejecutadas una sola vez antes del comienzo de la resolución del problema. También al mismo nivel que las tareas de preparación y las propias del CBR, se encuentras las tareas de mantenimiento, cuya ejecución tiene lugar después de todas las iteraciones del ciclo de ejecución.



Al mismo nivel que las tareas de preparación y las tareas del CBR, se encuentran las denominadas tareas de mantenimiento. Este tipo de tareas son resueltas tras las n iteraciones del ciclo de ejecución. Ejemplos típicos de esta clase de tarea son reorganizar la base de conocimiento o borrar los casos que no son útiles (estrategias de olvido).

El entorno jCOLIBRI proporciona una serie de métodos adicionales que permiten el tratamiento de las tareas de preparación y mantenimiento de los sistemas, conocidas como Pre-ciclo y Post-ciclo. Además, existe una biblioteca de métodos de resolución de problemas reusables destinadas a la resolución de dichas tareas.

Las descripciones de métodos en jCOLIBRI siguen una descripción elaborada que contiene los siguientes elementos:

- **Nombre.** Nombre completo de la clase que implementa el método. Esta clase debe implementar el interfaz "CBRMethod".

- **Descripción.** Descripción textual del método.

- **Precondición.** Descripción formal de los requisitos que se tienen que dar para que se pueda aplicar este método, incluyendo los referentes a los parámetros de entrada.

- **Tipo.** jCOLIBRI gestiona dos tipos de métodos: ejecución (o resolución) y descomposición. Los métodos de ejecución son aquellos que resuelven directamente la tarea para la cual han sido asignados, mientras que los de descomposición dividen la tarea a realizar en sub-tareas.

- **Parámetros.** Parámetros de configuración del método. Estos parámetros son los datos variables de la implementación del método. Por ejemplo, un método de recuperación puede tener como

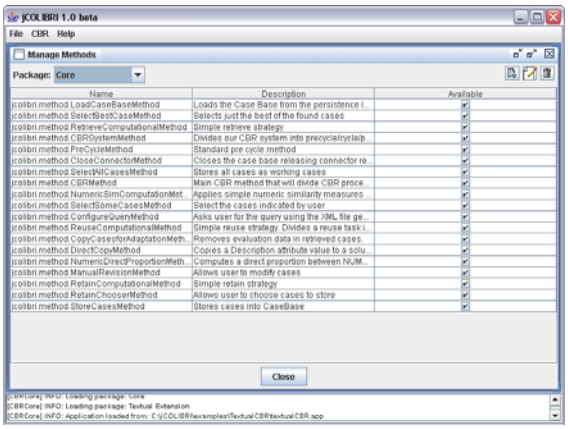
parámetro la función de similitud que ha de aplicar. Un parámetro puede ser cualquier tipo de objeto que implemente el interfaz "CBRTerm".

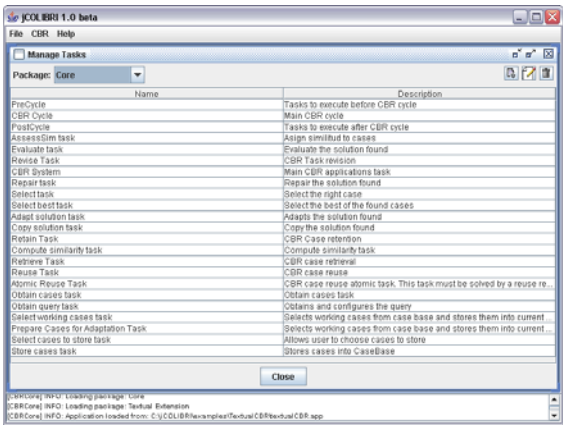
- **Competencias.** Lista de tareas que el método es capaz de resolver.

- **Sub-tareas.** Lista de tareas que resulta de dividir la tarea original en métodos de descomposición.

- **Poscondición.** Información de salida que se obtiene tras la ejecución del método. Estos datos serán utilizados para comprobar qué método puede tomar como entrada la salida de éste.

La interfaz gráfica de jCOLIBRI permite a sus usuarios crear y modificar nuevas tareas y métodos.





Bases de casos y Conectores

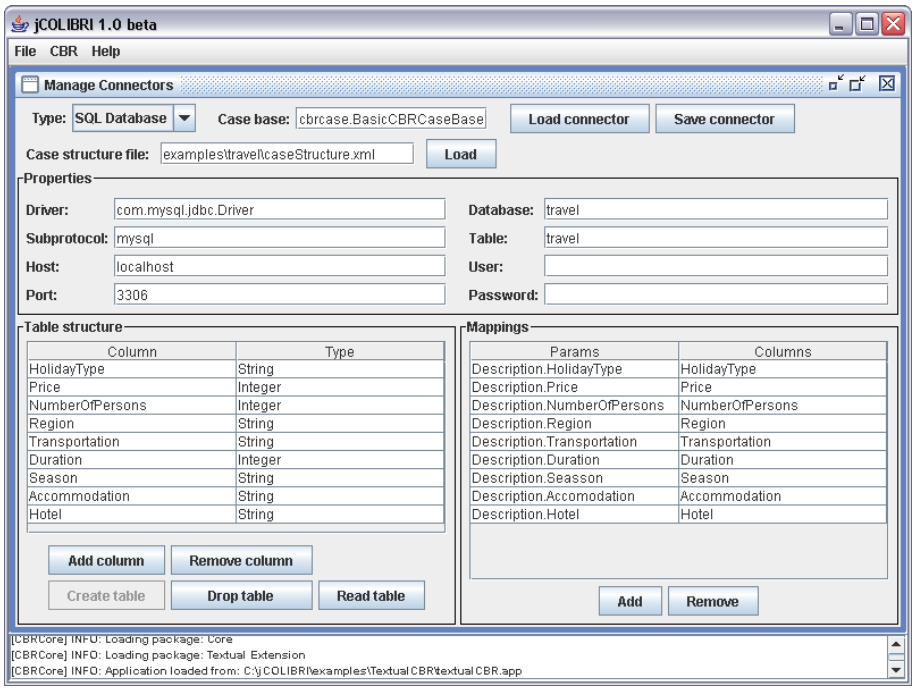
Los sistemas CBR deben poder acceder a los casos almacenados de una manera eficiente, un problema que se hace más relevante a medida que crece la base de conocimiento. El entorno jCOLIBRI divide la dificultad que presenta la gestión de la base de casos en dos diferentes, aunque relacionadas, vertientes: mecanismo de persistencia y organización en memoria.

La persistencia se construye mediante el uso de los conectores. Un conector representa la primera capa de jCOLIBRI referente al almacenamiento físico. Los conectores son objetos que conocen el mecanismo de acceso y recuperación de casos en el medio y cómo devolver éstos al sistema CBR de una forma uniforme. La utilización de conectores dota a jCOLIBRI de una flexibilidad respecto al almacenamiento físico que permite al diseñador elegir el más apropiado para sus necesidades.

El entorno jCOLIBRI incluye los siguientes conectores básicos:

- Pares atributo-valor separados por comas y almacenados en ficheros de texto planos.
- Casos almacenados en una o más tablas de una base de datos relacional.
- Casos formalizados mediante una lógica de descripción (Racer, Loom, OWL...).
- Casos almacenados en un fichero XML.
- Casos textuales almacenados en ficheros de texto.

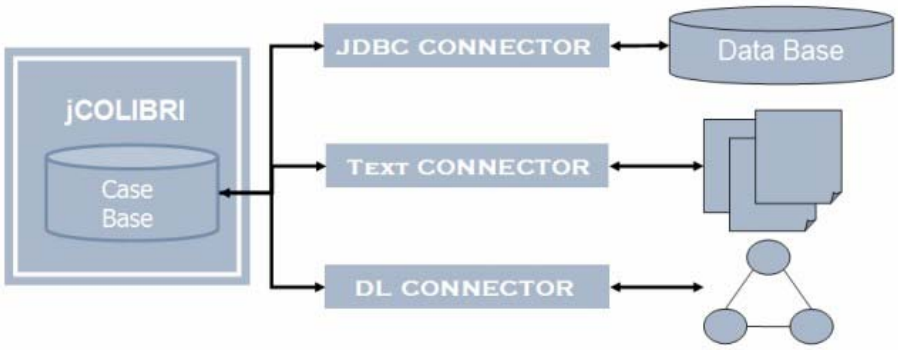
La interfaz gráfica de jCOLIBRI permite a los usuarios configurar los conectores:



El entorno jCOLIBRI incluye 'public abstract interface' en Java llamado "Connector", que pertenece al paquete "jcolibri.cbrcase", y en el que se definen una serie de métodos que deben ser implementados por cualquier conector. Concretamente, estos métodos son los siguientes:

- public void **init** (java.util.Properties). Inicializa el conector.
- public void **storeCases** (java.util.Collection). Almacena en el medio requerido la colección de casos que recibe como parámetro.
- public void **deleteCases** (java.util.Collection). Elimina un conjunto de casos de la persistencia.
- public Collection **retrieveAllCases**(). Recupera todos los casos que forman parte de la base de conocimiento.
- public Collection **retrieveSomeCases** (String). Recupera solo algunos de los casos de la base de casos, indicados mediante una expresión formalizada en pseudo-SQL.
- public void **close**(). Limpia cualquier recurso que se estuviera utilizando y suspende el servicio.

La segunda capa de la gestión de la base de conocimiento se refiere a la estructura de datos que es usada para organizar los casos una vez que son cargados en memoria. Esta organización (lineal, árboles k-d, redes de recuperación de casos...) pueden tener una gran influencia en los procesos CBR, por lo que el entorno deja a la elección del desarrollador la representación a utilizar.



Del mismo modo, la base de conocimiento también implementa un interfaz común que permite a los métodos CBR acceder a los casos que la componen. Las diferentes implementaciones de este interfaz deben proporcionar, al menos, las siguientes funcionalidades: construir una base de casos desde un medio de persistencia a través de un conector, aprender un caso, olvidar un caso y recuperar los k casos más parecidos a uno dado. Estos métodos están parametrizados con diversos datos como, por ejemplo, la query que permite acceder al conector o la función de similitud que se debe aplicar.

La organización en dos capas de la base de conocimiento es una alternativa muy potente que permite una gran cantidad de estrategias diferentes para acceder a los casos.

Tipos de datos

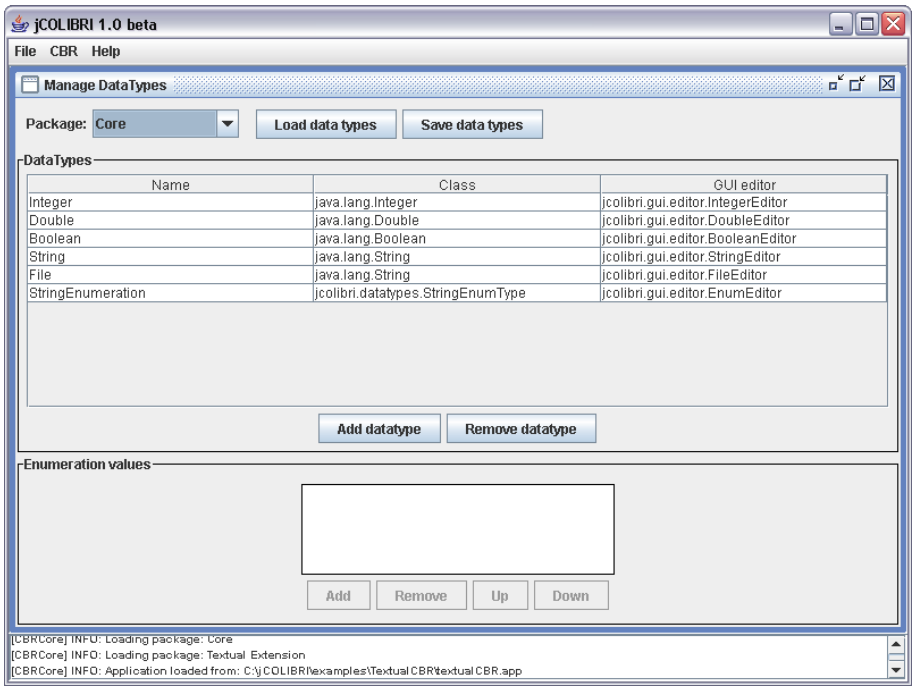
Un entorno CBR necesita llevar a cabo una gestión de tipos de datos por varios motivos:

- Para la creación de estructuras de casos.
- Para realizar consultas a los usuarios o para su utilización en los parámetros de los métodos.
- Para cargar/almacenar casos de/en la capa de persistencia.
- Para realizar el cómputo de similitudes entre atributos de un caso.

...

De esta forma, la versión actual del entorno jCOLIBRI incluye diversos ficheros de configuración que describen los tipos de datos que éste incluye y permite a los usuarios definir nuevos.

En jCOLIBRI los tipos de datos están compuestos por un nombre, por el objeto Java que contiene los datos y por el editor gráfico que pregunta a los usuarios por su valor.



Casos

jCOLIBRI representa los casos de una forma muy general. Un caso es, simplemente, un individual que puede tener cualquier número de relaciones con otros individuales (los atributos del caso).

El entorno contiene un determinado número de tipos de datos primitivos predefinidos que permiten la definición de cualquier caso simple. Este acuerdo, inspirado en el patrón de diseño de composición, proporciona una forma de tratamiento uniforme de cualquier estructura de casos compleja, siempre que ésta se ajuste al patrón de un individual ("Individual") que se relaciona con otros a través de diferentes relaciones ("IndividualRelation").

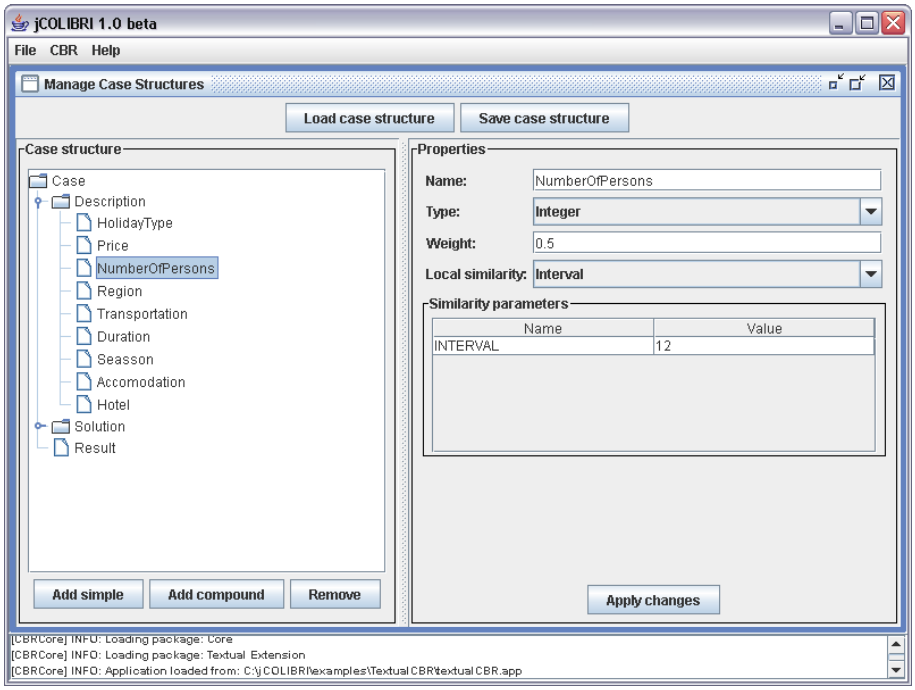
Para poder definir y manejar estructuras de casos, las herramientas de jCOLIBRI permiten a los usuarios configurar sus propios casos utilizando los tipos de datos cargados y las funciones de similitud. Un caso está compuesto por:

- **Descripción.** Descripción del problema por medio de varios atributos. Las consultas del sistema CBR ("queries") sólo se componen de una descripción.

- **Solución.** Contiene la descripción de la solución del caso. Existen múltiples representaciones de la solución de un caso: atributos simples, mapas conceptuales...

- **Resultado.** Representa el resultado de aplicar el caso a una situación real. Es, por tanto, que la solución puede ser o positiva o negativa para la resolución del problema.

Las herramientas de manejo de la estructura de casos representan los casos usando archivos XML que pueden ser intercambiados entre los usuarios. De esta forma, una estructura de casos puede ser utilizada por los métodos de resolución de problemas para la creación de nuevos casos y consultas o par poder modificar los casos que han sido cargados en la base de conocimiento.



Funciones de ayuda

Las funciones de ayuda completan los métodos de resolución de problemas. Existen varios tipos de estas funciones, que pueden ser dependientes del dominio. Entre todas, las más importantes son las funciones de similitud, aunque pueden existir otras como las funciones de adaptación.

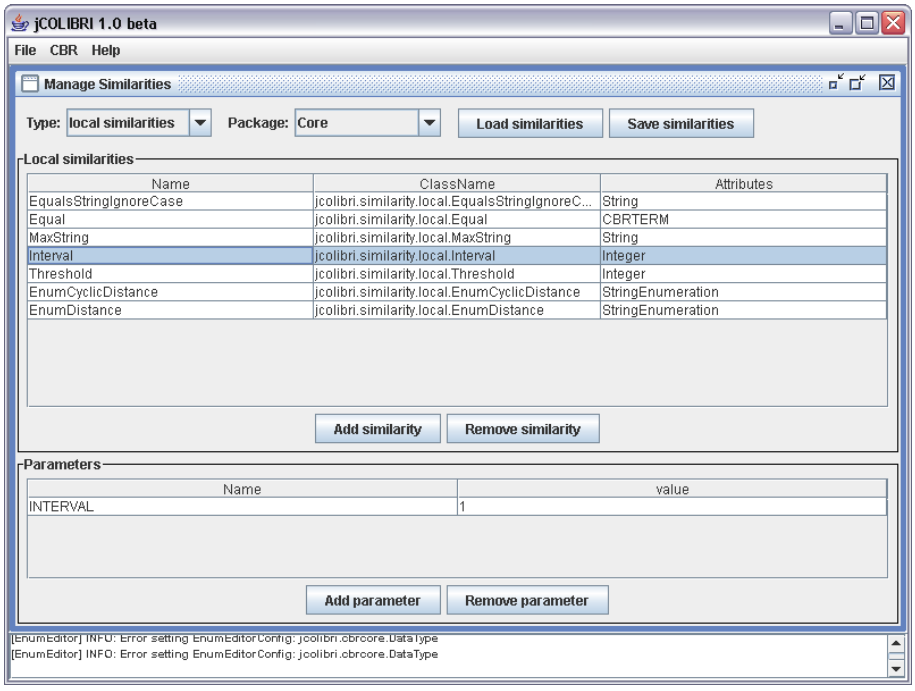
Las **Funciones de Similitud** calculan la similitud que existe entre casos (más concretamente, entre la query y un caso). De esta forma, el método de recuperación puede seleccionar aquellos casos de la base de conocimiento que más se parezcan a la query del sistema CBR.

Como el entorno jCOLIBRI utiliza estructuras de composición para la representación de casos, existen dos tipos de funciones de similitud:

- **Funciones de similitud local.** Calculan la similitud entre atributos simples. Se encuentran asociadas a tipos de datos específicos y pueden estar parametrizadas.

- **Funciones de similitud global.** Calculan la similitud entre atributos compuestos. Calculan, normalmente, la media de los valores obtenidos por las funciones de similitud local.

Las funciones de similitud se utilizan durante la creación de las estructuras de casos y las herramientas de jCOLIBRI permiten su configuración.



El núcleo (Core)

El "Core" (denominado "CBRCore" en el código fuente) es el componente más importante de todo el entorno. Se encarga del mantenimiento de la configuración del CBR y de la ejecución de la aplicación.

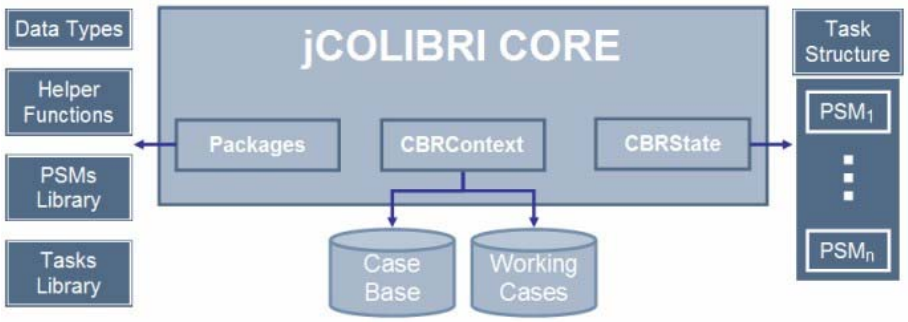
Cuando un usuario crea una plantilla de una aplicación CBR, en realidad lo que está haciendo es generar el código Java que configura un componente "Core" con las tareas, métodos, estructuras de casos, tipos de datos... apropiados. Después, para ejecutar dicha aplicación, el usuario sólo ha de llamar a los métodos del núcleo.

El núcleo está compuesto por los siguientes componentes:

- **CBRState.** Estado del CBR. Mantiene la configuración de los métodos y las tareas.

- **CBRContext.** Actúa como una pizarra donde los métodos pueden intercambiar datos. Suele contener la base de conocimiento y los casos con los que se está trabajando.

- **Paquetes.** Gestionan el resto de componentes: tipos de datos, funciones de similitud, estructuras de casos...



Extensiones

El entorno jCOLIBRI se ha diseñado siguiendo una estructura de extensiones que actúa como una arquitectura de plugins. Cada extensión contiene una serie de métodos, tareas, tipos de datos y funciones de ayuda. De esta forma, el sistema puede ser organizado y gestionado utilizando estos paquetes.

La versión actual de jCOLIBRI incluye las extensiones "core", "textual" y "usuario", que son estables, y los paquetes experimentales DLs y CRNs.

Los desarrolladores de jCOLIBRI pueden crear sus propias extensiones CBR y compartirlas para que puedan ser utilizadas por otras personas. Este proceso resulta muy sencillo ya que sólo es necesario crear un fichero de configuración XML y guardarlo bajo la carpeta "config". jCOLIBRI lee automáticamente este fichero y gestiona toda la información que es necesaria.

Apartado 4
Fases de desarrollo

Detalle de los puntos principales

En este cuarto apartado de la memoria se explican en detalle cada una de las etapas de desarrollo que se han llevado a cabo en el diseño de nuestro sistema de atención. En la primera de estas fases se realiza el diseño inicial de dicho sistema y en cada una de las siguientes se van añadiendo nuevas funcionalidades a éste. Este modelo de trabajo es conocido en Ingeniería del Software como "Prototipado Operacional": prototipo (modelo de sistema planificado) iterativo que es progresivamente refinado hasta que se convierte en el sistema final.

Las cuatro fases de investigación y desarrollo de nuestro proyecto han sido las siguientes:

- **Fase 1: Diseño inicial del sistema.** En esta primera fase de nuestro trabajo se lleva a cabo el diseño inicial de nuestro sistema, con la estructura y comportamiento definidos en la sección de "Requisitos y objetivos iniciales": un usuario envía una consulta textual al sistema, éste extrae los datos relevantes que la componen y utiliza el sistema CBR para obtener una solución con la que generar una respuesta que devolver.

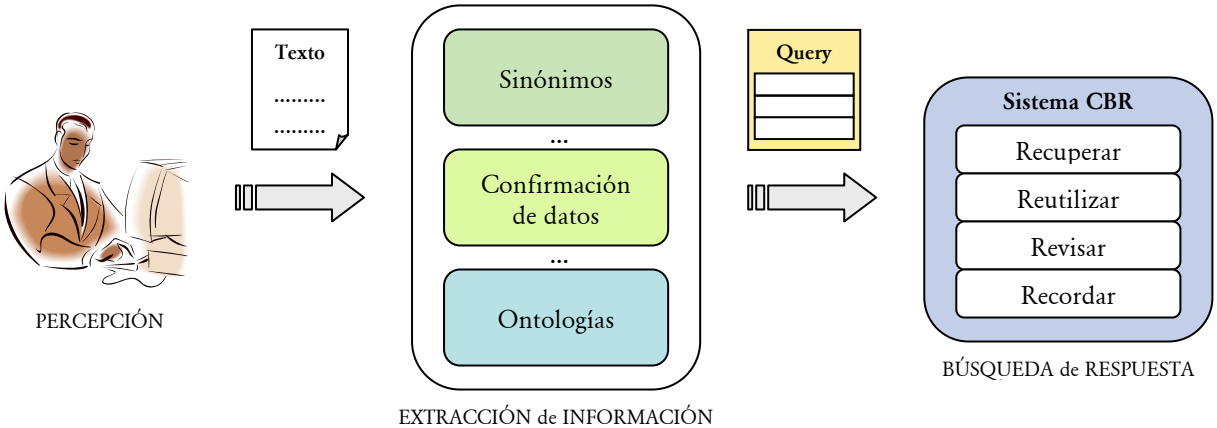
- **Fase 2: Utilización de sinónimos**. En esta segunda etapa de desarrollo se añade una nueva utilidad al sistema, consistente en la utilización de sinónimos en la etapa de Extracción de Información. Esta funcionalidad permite al sistema tener un mejor comportamiento frente a las consultas textuales de los usuarios.

- **Fase 3: Confirmación de información extraída.** En este fase se añade una nueva funcionalidad al sistema que permite mostrar al usuario los datos que han sido extraídos de su consulta De esta forma, es posible que éste confirme sus datos, con la opción de poderlos modificar si así lo considera oportuno.

- **Fase 4: Utilización de ontologías.** En la última fase de desarrollo de nuestro proyecto se lleva a cabo una nueva funcionalidad que permite al sistema, mediante la utilización de una ontología, obtener información relacionada con un cierto concepto buscado. De esta forma, se añade la capacidad de poder generar respuestas con datos similares

En los próximos apartados se explican en detalle cada una de estas etapas, describiendo, para cada una de ellas, el trabajo realizado, y especificando las utilidades añadidas y las limitaciones existentes. Además, se enumeran las dificultades encontradas a lo largo de este desarrollo y se realiza un estudio con los resultados obtenidos, de forma que se demuestren las nuevas características del sistema.

Es importante destacar que el trabajo realizado en cada una de estas etapas es complementario al anterior, de forma que cada una de ellas añade nuevas utilidades a la anterior. Así, para llevar a cabo una nueva aproximación es necesario utilizar la anterior para "construir" nuevas funcionalidades.



Fase 1: Diseño inicial del sistema

La finalidad de esta primera etapa de nuestro proyecto es la construcción inicial de nuestro sistema, en el que se permita a los usuarios enviar sus consultas, que éstas puedan ser analizadas y su información relevante extraída y que un sistema CBR busque una solución adecuada que pueda ser devuelta en forma de respuesta.

Como ya comentamos en el apartado de "Requisitos y objetivos iniciales" de este documento, el entorno jCOLIBRI permite la creación de sistemas CBR genéricos para cualquier tipo de dominio específico. Vamos a utilizar, por tanto, este módulo para llevar a cabo nuestra etapa de "Generación de Respuesta". Además, este entorno proporciona una extensión textual con un conjunto de métodos para realizar el procesamiento de textos y extracción de información sobre éstos, así como infraestructura para poder representar este tipo de datos.

El diseño de sistemas en este entorno se lleva a cabo mediante la división en distintas etapas del trabajo a realizar, entre las que se incluirán, por supuesto, las propias del razonamiento basado en casos (ciclo CBR): Recuperar, Reutilizar, Revisar y Recordar (las 4 R's). Pero además se deben de incluir todas las tareas relativas al comportamiento de nuestro sistema: percepción y extracción de información. Según ya se ha explicado en apartado anteriores, estas dos tareas tienen la finalidad de recibir las consultas de los usuarios y convertirlas de textos a representaciones estructuradas de datos relevantes entendibles por las etapas del CBR (en forma de caso), por lo que han de ser previas a éstas.

En el entorno jCOLIBRI, los sistemas están divididos en tareas y sub-tareas, agrupadas en tres etapas principales presentes en todos:

- **Pre-ciclo.** En esta etapa se realizan todas las tareas previas al ciclo CBR, como, por ejemplo, conectar con una base de datos para obtener todos los datos que formarán parte de la base de casos.

- **Ciclo CBR.** Ésta es la etapa principal del sistema, en la que se llevan a cabo las acciones principales, incluidas las cuatro tareas pertenecientes al razonamiento basado en casos (4 R's). Además, en esta etapa se pueden incluir más operaciones como, por ejemplo, mostrar a los usuarios una ventana en la que pueden introducir los datos de entrada (área de texto para escribir una consulta textual, formulario de selección de valores...).

- **Post-ciclo.** En esta última etapa del sistema se han de realizar tareas posteriores al ciclo CBR y que deben ser llevadas a cabo antes de que se termine la ejecución del sistema, como, por ejemplo cerrar un conector con una base de datos.

Una vez introducidos los conceptos básicos de los sistemas CBR en el entorno jCOLIBRI, el primer paso para en nuestro trabajo es definir el comportamiento que, en esta primera etapa ha de tener nuestro sistema, así como su interacción con los usuarios. En el segundo apartado de este documento se decidió que el dominio a utilizar durante el desarrollo de nuestro trabajo sería el de los viajes. Sabiendo ésto, podemos definir el comportamiento general de nuestro sistema de la siguiente forma:

- Un usuario envía al sistema una consulta en forma de texto en la que describe el tipo de viaje que desea realizar, con toda la información que desee incluir.
- El sistema analiza la consulta del usuario y extrae toda la información relevante.
- El sistema devuelve al usuario el caso o casos con mayor similitud con los datos de su consulta, es decir, el viaje o viajes que más se ajustan a sus preferencias.

Una vez pensado el comportamiento general que queremos que tenga nuestro sistema, es necesario definir cada una de las tareas que lo compondrán y que serán las encargadas de realizar todo el trabajo requerido. Cada una de estas tareas puede estar, a su vez, dividida en otras sub-tareas (tareas de descomposición) que realicen acciones más específicas, o resolver directamente el problema (tareas de resolución). En el entorno jCOLIBRI, las etapas básicas principales de los sistemas con razonamiento basado en casos quedan definidas de forma automática:

- Obtención de la base de casos.
- Obtención de la query del sistema (o consulta del usuario).
- Etapa de "Recuperación" del CBR.
- Etapa de "Reutilización" del CBR.
- Etapa de "Revisión" del CBR.
- Etapa de "Recuerdo" del CBR.

El siguiente paso es definir el comportamiento que tendrán cada una de estar tareas para nuestro sistema, es decir, especificar las tareas que llevarán a cabo cada una de ellas, teniendo en cuenta la definición que hemos hecho sobre el comportamiento general del sistema. Consideremos la siguiente definición:

- **Obtención de la base de casos.** En este caso sólo es necesario definir una única tarea que sea la encargada de obtener toda la información que formará parte de la base de conocimiento del sistema CBR y que servirá para encontrar la respuesta más adecuada a la consulta del usuario.

- **Obtención de la query del sistema.** La query del sistema se trata de un componente que tiene la misma estructura que cada uno de los casos de la base de conocimiento del sistema CBR, pero con los datos introducidos por el usuario. Al tener la misma estructura que todos los casos de la base, el sistema puede aplicar las funciones de similitud sobre ellos para poder obtener el que más se parezca con la consulta. Como el CBR necesita como query un componente con dicha estructura, será necesario transformar los datos de entrada de nuestro sistema, en forma de texto, a una representación estructurada que contenga los valores relevantes descritos por el usuario en ésta. Para ello, será necesario definir en esta etapa las siguientes tareas:

  - Una primera tarea encargada de mostrar al usuario una ventana donde pueda escribir su consulta textual.
  - Una tarea de extracción de información, que analice la consulta textual del usuario y extraiga los datos relevantes de ésta.
  - Una tarea que se encargue de crear una query aceptada por el sistema, es decir, con la estructura de casos correspondiente, y que contenga la información extraída en la etapa anterior.

- **Etapa de "Recuperación".** Esta es la primera etapa del ciclo CBR. En ella se deben de aplicar las funciones de similitud entre la query del sistema y cada uno de los casos de la base de conocimiento, para así encontrar el más parecido y devolverlo como respuesta a la consulta del usuario.

- **Etapas de "Reutilización", "Revisión" y "Recuerdo".** Como la finalidad de nuestro sistema es, por el momento, encontrar el caso con una mayor similitud a la consulta del usuario, estas tres etapas no serán utilizadas en estas primeras aproximaciones del proyecto, aunque siempre se deja como alternativa para un futuro.

Como ya hemos dicho antes, es obligatorio incluir en nuestro sistema las tres etapas principales de Pre-ciclo, Ciclo CBR y Post-ciclo. Por tanto, es necesario distribuir cada una de las tareas anteriores entre estas tres etapas de la siguiente forma:

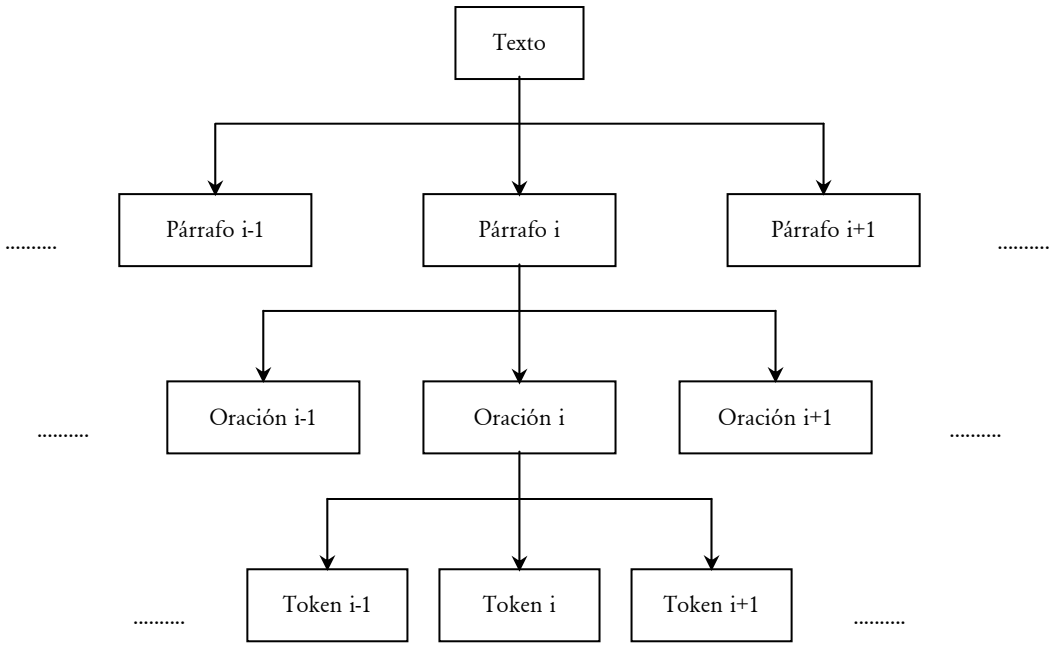
- Pre-ciclo
  - Obtención de la base de casos.

- Ciclo CBR
  - Obtención de la query del sistema.
  - Etapa de "Recuperación" del CBR.
  - Etapa de "Reutilización" del CBR (no se va a utilizar).
  - Etapa de "Revisión" del CBR (no se va a utilizar).
  - Etapa de "Recuerdo" del CBR (no se va a utilizar).

- Post-ciclo

En la siguiente sección de este apartado se detalla por completo la estructura del sistema y las etapas en las que se divide cada uno de los puntos anteriores, así como las acciones que éstas realizan.

Para llevar a cabo el tratamiento de textos en nuestro sistema se va a utilizar la **extensión textual** que nos proporciona el entorno jCOLIBRI. Esta extensión provee una infraestructura de clases, con atributos y métodos, que permiten la representación de textos para que éstos puedan ser tratados de forma correcta por el resto de utilidades del entorno. En el entorno de jCOLIBRI, un texto no es simplemente un conjunto de cadenas de caracteres, sino que es una estructura en la que es posible almacenar una gran cantidad de información. La representación de esta estructura es la siguiente:



Esta descomposición considera la descomposición de textos en párrafos, éstos en oraciones y éstas, por último, en tokens. A continuación se detallan todos los aspectos de cada una de estas estructuras:

- **Texto (Text).** Representa el texto completo, compuesto por una serie de párrafos. Un texto en el entorno jCOLIBRI contiene la siguiente información:

- PARAGRAPHS: conjunto de párrafos que forman el texto.

- **Párrafo (Paragraph).** Representa un párrafo perteneciente a un determinado texto. Un párrafo en jCOLIBRI permite representar la siguiente información:

  - RAW_DATA. Atributo donde es almacenado el fragmento de texto original del párrafo.
  - SENTENCES. Conjunto de oraciones que forman el párrafo.
  - FEATURES. Conjunto de elementos "FeatureInfo" (característica del texto) extraídos por un método de extracción de características a partir del fragmento de texto correspondiente a este párrafo.
  - PHRASES. Conjunto de elementos "PhraseInfo" (expresión propia del dominio) extraídos por un método de identificación de expresiones propias a partir del fragmento de texto correspondiente a este párrafo.
  - TOPICS. Conjunto de cadenas de texto que representan los temas extraídos por un método de clasificación de temas a partir del fragmento de texto correspondiente a este párrafo.

- **Oración (Sentence).** Representa a una oración perteneciente a un determinado párrafo. Una oración en jCOLIBRI permite representar los siguientes datos:

  - RAW_DATA. Atributo donde se almacena el fragmento de texto original de la oración.
  - SENTENCE_POSITION. Indica la posición de esta oración entre todas las del párrafo correspondiente.
  - TOKENS. Conjunto de tokens que forman la oración.

- **Token (Token).** Los tokens son la unidad mínima dentro de un texto. Un token es una palabra más toda la información acerca de ella. En este entorno permite representar los siguientes datos:

  - COMPLETEWORD. Contiene la palabra completa.
  - STEMMEDWORD. Contiene la raíz de la palabra, almacenada aquí por un método de extracción de raíces.
  - POSTAG. Contiene la categoría léxica de la palabra .Este datos es almacenado aquí por un método de etiquetado de categorías léxicas.
  - TOKENINDEX. Indica la posición del token entre todas las de la oración a la que pertenece.
  - WORDPOSITION. Posición de la palabra dentro del texto original de su párrafo.
  - RELATEDTOKENS. Contiene una lista con pesos de tokens relacionados.
  - ISNAME. Atributo de tipo booleano que indica si la palabra es o no un nombre. Este dato es almacenado aquí por un método de extracción de nombres.
  - ISNOTSTOPWORD. Atributo de tipo booleano que indica si se trata de una palabra de parada. Este dato es almacenado aquí por un método de filtro de palabras.

Las oraciones y los párrafos son utilizados para estructurar los datos a modo de contenedores. Los textos son contendores finales y constituyen por sí mismos un tipo de datos, por lo que se podrán declarar atributos de tipo "Text".

Además de estas representaciones textuales, jCOLIBRI incluye una serie de clases ya implementadas con operaciones para el procesamiento y análisis de textos, que nosotros utilizaremos en la etapa de extracción de información para poder pasar de una representación textual de la consulta del usuario a una en forma de caso. Estas operaciones provistas por jCOLIBRI son:

- **Filtro de palabras** (Words Filter). Filtra y convierte en tokens un texto quitando las palabras de parada (stop-words) y los caracteres especiales.

- **Etiquetado de categorías léxicas** (Part-of-speech Tagging). Este método utilizada un etiquetador de Máxima Entropía (implementado por OpenNLP) para etiquetar palabras con sus correspondientes categorías léxicas.

- **Algoritmos de extracción de raíces** (Stemmer Algorithms). Este método permite llevar a cabo diversos métodos de extracción de raíces en varios idiomas. Para ello, utiliza el paquete externo "Snowball", que también define un lenguaje de extracción de raíces para permitir parsear otros lenguajes.

- **Extracción de nombres** (Name Extraction). Selecciona los principales nombres del texto utilizando para ello un algoritmo de Máxima Entropía.

- **Identificación de expresiones propias** (Phrase Identification). Método que extrae las expresiones propias de un dominio específico mediante el uso de expresiones regulares. Un desarrollador puede definir una expresión propia utilizando un fichero de configuración como este:

```
# Formato de reglas: [FeatureName]FeatureRegularExpresion
[Compaq Presario 2100](Compaq|HP|Hewlett-Packard)? Presario 2100
```

  En esta regla de ejemplo encuentra el mismo concepto (un modelo de ordenador) que puede ser escrito utilizando distintas formas. La sintaxis concreta para definir cada regla está descrita en el fichero de configuración. Este método es dependiente del dominio.

- **Glosario** (Glossary). Relaciona las palabras de una query con las palabras de los casos de la base de conocimiento utilizando para ello un glosario específico del dominio en el que se trabaje. También utiliza un fichero de configuración donde los desarrolladores pueden definir la similitud entre palabras:

```
# Formato de reglas: [Part-of-Speech Tag]{Similarity} word1 word2 ... wordn
[NOUN]{2} case instance
[NOUN]{3} cbr nbr
[NOUN]{1} word term speech
```

  En este caso, el desarrollador debe especificar la categoría léxica de la palabra (en inglés), porque una misma palabra puede tener distintos significados dependiendo si es utilizado como verbo, nombre, adjetivo, etc. Estas reglas también permiten la definición de tres niveles de similitud entre palabras (1 es el mayor y 3 el menor). Este método es dependiente del dominio.

- **Sinónimos** (Thesaurus). Este método relaciona las palabras que forman la query del sistema con las de los casos de la base de conocimiento, utilizando para ello WordNet. Su finalidad es mejorar el grado de coincidencia entre query y casos incluyendo las relaciones entre palabras. Para su implementación se ha utilizado un paquete externo llamado "JWordNetLibrary" (JWNL), donde se define un simple API de utilización de WordNet.

- **Extracción de características** (Feature Extraction). Extrae características de un texto utilizando expresiones regulares y las almacena como pares atributo-valor:

```
# Formato de reglas: [FeatureName]{FeaturePosition}FeatureRegularExpresion
# FeatureName es utilizado para almacenar la información extraída
[Person]{2}(Mr.|Mss.) ((\p{Lu}(\w+|\.)\s)+)
[University]{1}((\p{Lu}\w+\s)+)University
[Company]{1}((\p{Lu}\w+\s)+)(Inc\.|Corporation|Associates|Bank)
```

  Estos ejemplos extraen características ("Features") mediante la identificación de estructuras típicas: un nombre de persona está precedido por Mr (Sr.) o Mrs (Sra.), el nombre de una

universidad va seguido de la palabra "University" (Universidad)... Este método es dependiente del dominio.

- **Clasificación por temas** (Topic Classification). Este método asocia un tema ("Topic") utilizando para ello características y expresiones propias extraídas por los métodos anteriores. Esta capa define una descripción de alto nivel del texto que puede resultar muy útil para tareas de indexación. Utiliza un fichero de configuración de la forma:

```
# Formato de reglas: [Topic] <FeatureName,Value> <FeatureName,Value>…
<Phrase> <Phrase>
# Topic: clasificación por tema
# FeatureName: "FeatureName" definido en la capa de extracción de
# características.
# Value: valor del "FeatureName". También puede ser '?', significando
# cualquier valor.
# Phrase: es posible utilizar las expresiones propias detectadas en la capa
# correspondiente.
[MyProject]<University,Complutense de Madrid> <Person,Juan Antonio>
<Company,?> <Compaq Presario 2100>
```

  Este ejemplo muestra la descripción de un tema. Un texto acerca de un proyecto ("MyProject") debería tener características acerca de una universidad ("University"), una persona ("Person") y el ordenador de esa persona ("Compaq Presario 2100"). Este método es dependiente del dominio.

- **Extracción de información básica** (Basic Information Extraction). Extrae información de textos y la almacena en atributos de casos (si están definidos). Si el procesamiento textual encuentra una característica con una etiqueta que coincide con otra de un determinado caso, entonces este método copiará el contenido de la característica (su valor) al caso.

Por último, es necesario definir la estructura que tendrán cada uno de los casos de la base de conocimiento de nuestro sistema CBR. En el apartado "Requisitos y objetivos iniciales" de este documento ya se enumeraron todos los atributos en los que se dividía cada uno de los viajes que forman parte de la base de casos. Para definir la estructura de la que hablamos es necesario asignar a cada atributo existente un tipo y la función de similitud que se utilizará para poder ser comparado con el mismo atributo de otro caso. Para nuestro dominio de viajes la estructura de casos queda de la siguiente forma:

- **HolidayType**
  Tipo: String
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = HolidayTypesEnum
  Peso: 1.0

- **Price**
  Tipo: String
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 2500
  Peso: 1.0

- **NumberOfPersons**
  Tipo: String
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 12
  Peso: 0.5

- **Region**
  Tipo: String
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = RegionsEnum
  Peso: 1.0

- **Transportation**
  Tipo: String
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = TransportationsEnum
  Peso: 0.7

- **Duration**
  Tipo: String
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 21
  Peso: 0.5

- **Season**
  Tipo: String
  Función de similitud local: MinEnumCyclicDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = SeasonsEnum
  Peso: 0.7

- **Accommodation**
  Tipo: String
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = AccomodationsEnum
  Peso: 0.7

- **Hotel**
  Tipo: String
  Función de similitud local: EqualsStringIgnoreCase
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 0.3

  Función de similitud global: **Average**.

El significado de cada uno de estos campos es el siguiente:

- **Tipo.** Corresponde al tipo del atributo. Puede ser un tipo básico (String, Integer...) o uno definido por el desarrollador. El motivo por el que en esta primera aproximación todos los atributos son de tipo cadena de texto es porque en ellos se van a almacenar valores copiados del texto de consulta del usuario. Si es necesario guardar en estos atributos más de un valor, éstos serán separados por espacios en blanco entre cada dos. Debido a esta representación, las funciones de similitud local tendrán que adaptarse a este formato para poder trabajar con estos datos. En el próximo apartado se detalla más acerca de todo esto.

- **Función de similitud local.** Se trata del nombre de la función de similitud local que será utilizada para comparar los valores del atributo correspondiente de dos casos diferentes (generalmente, la query del sistema y cada uno de los casos de la base de conocimiento). Estas funciones han de ser definidas en el entorno jCOLIBRI e implementadas mediante un método Java. Cada una de ellas devuelve un valor entre 0 y 1 indicando la similitud entre los dos valores comparados, siendo 1 la igualdad entre ambos. Algunas de estas funciones

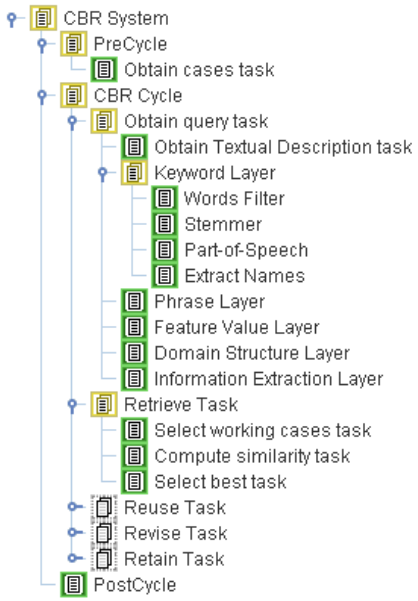
requieren el uso de un parámetro para calcular su resultado. En el próximo apartado se explican en detalle cada una de estas funciones.

- **Peso.** Este campo indica la importancia que tiene cada atributo, siendo 1 el máximo valor y 0 el mínimo. Es utilizado para ponderar cada uno de los atributos dentro del cálculo de similitud global entre dos casos.

- **Función de similitud global.** Se trata de la función que será aplicada para calcular la similitud global entre dos casos, es decir, cuánto se parece un caso a otro, mediante un resultado numérico de 0 a 1. Para calcular este resultado, la función utiliza todos los valores de similitud local obtenidos para cada uno de los atributos que compone el caso, aplicando sobre ellos su algoritmo. También ha de utilizar los pesos definidos para ponderar cada uno de estos valores. En el caso del ejemplo, la función "Average" calcula la media de todos los valores de similitud local, ponderados por sus respectivos pesos.

Etapas del sistema

En la siguiente figura se muestra la estructura de etapas en el entorno jCOLIBRI para esta primera fase del diseño inicial de nuestro sistema. En ella se puede observar que existen, como ya hemos explicado en el apartado anterior, tres capas principales: Pre-ciclo, Ciclo CBR y Post-ciclo. Cada una de ellas se descompone en una serie de sub-tareas más específicas que realizan parte del trabajo.



A continuación se enumeran y describen brevemente cada una de las etapas de nuestro sistema, a modo de resumen e introducción para una posterior explicación en detalle. En esta primera fase se van a explicar cada una de las etapas que forman parte del sistema, pero en las posteriores sólo se hará mención sobre las de nueva adición.

- **Pre-ciclo** (PreCycle).
    - **Tarea de obtención de casos** (Obtain cases task). En esta tarea se obtiene la base de conocimiento del sistema CBR, mediante la conexión con una base de datos que almacena cada uno de los valores que forman parte de los diferentes casos disponibles.
- **Ciclo CBR** (CBR Cycle).
    - **Tarea de obtención de la query** (Obtain Query Task). Esta tarea divide su trabajo en una serie de sub-tareas, con el objetivo de obtener la query del sistema CBR, es decir, una representación de los datos del usuario con la misma estructura que cada uno de los casos de la base de conocimiento.
        - **Tarea de obtención de la consulta textual** (Obtain Textual Description task). Esta tarea se encarga de mostrar a los usuarios del sistema una pequeña ventana en la que pueden escribir sus consultas textuales que después serán analizadas.
        - **Capa de palabras clave** (Keyword Layer). Capa de descomposición en la que se analizan cada una de las palabras que forman la consulta del usuario.

- **Filtro de palabras** (Words Filter). Filtra y convierte en tokens un texto quitando las palabras de parada (stop-words) y los caracteres especiales.

- **Extracción de raíces** (Stemmer). Extrae las raíces de todas las palabras obtenidas en la etapa anterior.

- **Categorías léxicas** (Part-of-Speech). Etiqueta cada una de las palabras anteriores con sus correspondientes categorías léxicas (nombres...).

- **Extracción de nombres** (Extract Names). Selecciona y extrae los principales nombres que forman parte del texto de consulta.

- **Capa de expresiones propias** (Phrase Layer). Método que extrae las expresiones propias de un dominio específico mediante el uso de expresiones regulares. Cada desarrollador puede definir una serie de expresiones propias mediante reglas en un fichero de configuración

- **Capa de características** (Feature Value Layer). Extrae características de un texto utilizando expresiones regulares y las almacena como pares atributo-valor.

- **Capa de estructura de dominio** (Domain Structure Layer). Este método asocia temas a diferentes partes del texto de consulta, utilizando para ello características y expresiones propias extraídas por los métodos anteriores.

- **Capa de extracción de información** (Information Extraction Layer). Almacena la información extraída por las capas anteriores en una estructura de casos que se convertirá en la query del sistema CBR.

- **Tarea Recuperar** (Retrieve Task). Primera etapa de un ciclo CBR, que tiene como objetivo la recuperación de un caso o varios de la base de conocimiento mediante su comparación con la query obtenida en la capa anterior. Divide su trabajo en varias sub-tareas.

  - **Tarea de selección de casos activos** (Select working cases task). El sistema selecciona cuáles serán los casos que van a intervenir en la tarea de recuperación. En nuestro caso, y generalmente siempre, se seleccionan todos los que forman parte de la base de conocimiento.

  - **Tarea de cómputo de similitud** (Compute similarity task). En esta tarea se aplican las funciones de similitud definidas para el sistema con el fin de obtener datos numéricos de similitud entre la query y cada uno de los casos de trabajo.

  - **Tarea de selección del mejor** (Select best task). En esta capa el sistema CBR recupera el caso o casos con mayor similitud con la query. En nuestro caso, se ha especificado que sólo se seleccione el mejor (el que más parecido tenga).

- **Tarea Reutilizar** (Reuse Task). Esta tarea no se utiliza.
- **Tarea Revisar** (Revise Task). Esta tarea no se utiliza.
- **Tarea Recordar** (Retain Task). Esta tarea no se utiliza.

- **Post-ciclo** (PostCycle). La única acción que se realiza en esta etapa es cerrar el conector que se utilizo en el Pre-Ciclo para obtener la base de conocimiento del sistema CBR.

Pre-ciclo (PreCycle)

Localización: PreCycle (etapa principal).
Clase Java que implementa la tarea: jcolibri.method.PreCycleMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

El Pre-ciclo es una tarea de descomposición en la que se obtienen los casos de la base de casos.

Tarea de obtención de casos (Obtain cases task)

Localización: PreCycle → Obtain cases task.
Clase Java que implementa la tarea: jcolibri.method.LoadCaseBaseMethod.
Parámetros: fichero de configuración del conector (de tipo File).
Tipo de tarea: resolución.

La primera acción que es necesario realizar en nuestro sistema es la obtención de la base de casos. Como ya hemos comentado anteriormente, en el caso del dominio de los viajes los datos se encuentran almacenados en una base de datos. El entorno jCOLIBRI proporciona métodos para la extracción de estos valores, mediante conectores, y su incorporación al sistema de razonamiento basado en casos.
Para ello, lo primero es definir la estructura que va a tener cada uno de los casos de la base de casos y que, evidentemente, coincide con la realizada en la base de datos para almacenar los valores. Por tanto, lo único que nosotros tenemos que hacer es indicarle al conector en que atributos de la estructura del caso debe guardar cada uno de los valores extraídos de la base de datos.

De esta forma, el conector extraerá de la base de datos todos los valores que ahí encuentre e irá creando uno a uno los casos que pasarán a formar parte de la base de casos.

Ciclo CBR (CBR Cycle)

Localización: CBR Cycle (etapa principal).
Clase Java que implementa la tarea: jcolibri.method.CBRMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

El Ciclo CBR es una tarea de descomposición en la que se obtiene la consulta del usuario y se ejecutan las cuatro etapas típicas de un sistema CBR: recuperar, reutilizar, revisar y recordar. En el caso del dominio de los viajes, nuestro objetivo es sólo llevar a cabo la tarea de recuperación, para poder contestar al usuario con un viaje lo más acorde a sus preferencias.

Tarea de obtención de la query (Obtain Query Task)

Localización: CBR Cycle → Obtain Query Task.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.ConfigureTextualQueryMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

Esta tarea descompone su trabajo en varias etapas para poder obtener la consulta textual del usuario. Su primera misión será recoger la descripción del viaje del usuario en forma de texto. Después, se encargará de analizar esta consulta y de extraer toda la información posible. Por último, construirá una query con

la misma estructura que el resto de casos de la base de casos, para dejar preparado el trabajo a la siguiente etapa del sistema (recuperación de casos).

Tarea de obtención de la consulta textual (Obtain Textual Description Task)

Localización: CBR Cycle → Obtain Query Task → Obtain Textual Description Task.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.ObtainTextualDescriptionMethod.
Parámetros: ninguno.
Tipo de tarea: resolución.

Esta tarea es la encargada de mostrar al usuario del sistema una pequeña ventana con un área de texto en la que puede escribir la descripción del viaje de desea realizar. Esta ventana contiene además dos botones, uno para aceptar la acción y proceder a las siguientes etapas, y otro para cancelar y salir de la aplicación.

Cuando el usuario escribe su descripción textual y pulsa el botón de aceptar, esta tarea se encarga de crear una query especial (CBRTextualQuery), que contiene las características generales de una query genérica (CBRGenericQuery) pero que contiene un solo atributo de tipo texto (Text), en el que se almacena la consulta del usuario.

Una vez creada esta query, se añadirá al contexto del sistema, componente que sirve de comunicación entre las ejecuciones de las distintas tareas que lo conforman. De hecho, el contexto mantiene el estado actual del sistema CBR en todo momento.

Capa de palabras clave (Keyword Layer)

Localización: CBR Cycle → Obtain Query Task → Keyword Layer.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.KeywordLayerMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

Esta capa se encarga de realizar un análisis léxico del la descripción textual que el usuario introdujo en la anterior etapa del sistema. Para empezar, se reconocen las palabras simples que forman parte del texto y se quitan las de parada (stop-words). Además, se extraen los nombres del texto y, por último, se etiqueta cada palabra con la categoría léxica a la que corresponde (nombre, adjetivo, verbo...).

Filtro de palabras (Words Filter)

Localización: CBR Cycle → Obtain Query Task → Keyword Layer → Words Filter.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.WordsFilterMethod.
Parámetros: procesar query (de tipo booleano) y procesar casos (de tipo booleano).
Tipo de tarea: resolución.

Esta tarea actúa de filtro con la consulta textual que el usuario escribió en la etapa anterior, separando el texto en palabras y eliminando las denominadas "stop-words" o palabras de parada, es decir, palabras que no tienen ningún valor semántico dentro el texto y no proporcionan información útil.

Esta tarea se divide en cuatro etapas:

- **Detectar oraciones.** Se encarga de detectar las oraciones que componen el texto.

- **Convertir oraciones en tokens.** Divide cada una de las oraciones anteriores en palabras simples y las almacena la información pertinente en los atributos "CompleteWord" y "TokenIndex" de su token correspondiente.

- **Filtrar palabras de parada.** Localiza en el texto las palabras de parada y rellena el campo booleano "IsNotStopWord" de su token correspondiente, indicando si lo es o no.

- **Crear índices.** Enlaza los tokens con el fragmento de texto con el que se corresponden, almacenando esta información en su atributo "WordPosition".

Extracción de raíces (Stemmer)

Localización: CBR Cycle → Obtain Query Task → Keyword Layer → Stemmer.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.StemmerMethod.
Parámetros: procesar query (de tipo booleano) y procesar casos (de tipo booleano).
Tipo de tarea: resolución.

Esta tarea se encarga de extraer las raíces de las palabras que conforman el texto introducido por el usuario y las almacena el atributo "StemmedWord" de su token correspondiente.

Categorías léxicas (Part-of-Speech)

Localización: CBR Cycle → Obtain Query Task → Keyword Layer → Part-of-Speech.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.PartofSpeechMethod.
Parámetros: procesar query (de tipo booleano) y procesar casos (de tipo booleano).
Tipo de tarea: resolución.

El objetivo de esta tarea es obtener para cada palabra que forma parte de la descripción textual del usuario, y que no sea palabra de parada, su categoría léxica. Ejemplos de estas categorías son nombre común, nombre propio, adjetivo comparativo, interjección, verbo en forma pasada, etc.

La categoría léxica de cada una de las palabras será almacenada en el atributo "POSTag" de su token correspondiente.

Extracción de nombres (Extract Names)

Localización: CBR Cycle → Obtain Query Task → Keyword Layer → Extract Names.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.ExtractNamesMethod.
Parámetros: procesar query (de tipo booleano) y procesar casos (de tipo booleano).
Tipo de tarea: resolución.

Esta tarea se encarga de localizar las palabras que son nombres dentro del texto del usuario, almacenando esta información en el atributo "IsName" de cada token correspondiente.

Capa de expresiones propias (Phrase Layer)

Localización: CBR Cycle → Obtain Query Task → Phrase Layer.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.ExtractPhrasesMethod.
Parámetros: procesar query (de tipo booleano), procesar casos (de tipo booleano) y fichero donde el usuario ha definido las expresiones específicas del dominio (de tipo File).

Tipo de tarea: resolución.

El objetivo de esta tarea es reconocer expresiones específicas del dominio en el que se está trabajando. Estas expresiones son definidas por el usuario del sistema en un fichero externo a la aplicación, y tienen el siguiente formato:

[FeatureName]FeatureRegularExpresion

- FeatureName: es el nombre del atributo de la estructura del caso donde se ha de almacenar la información extraída.
- FeatureRegularExpression: expresión definida siguiendo la sintaxis que se especifica en la clase Java java.util.regex.Pattern.

Ejemplo: [Compaq Presario 2100](Compaq|HP|Hewlett-Packard)? Presario 2100

El ejemplo anterior se interpreta de la siguiente forma: si en el texto introducido por el usuario se encuentra la expresión "(Compaq|HP|Hewlett-Packard)? Presario 2100" (ver su significado en el API de Java), en el atributo "Compaq Presario 2100" de la estructura del caso se almacenará el valor cierto.

Este tipo de reglas son útiles para los atributos de tipo booleano de la estructura del caso, ya que buscan en el texto del usuario si se encuentra alguna expresión específica del dominio. En caso de que sí se encuentre, el valor tomado por ese atributo será de cierto.
Las expresiones específicas del dominio encontradas en el texto serán almacenadas de forma provisional como parte de él, en atributos especiales encargados de almacenarlas, para que en la capa de extracción de información se copien a los atributos correspondientes de la estructura del caso.

En el caso particular del dominio de los viajes, no se ha introducido ninguna regla para esta etapa de análisis, ya que en la estructura de cada caso no hay ningún atributo de tipo booleano.

Capa de características (Feature Value Layer)

Localización: CBR Cycle → Obtain Query Task → Feature Value Layer.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.ExtractFeaturesMethod.
Parámetros: procesar query (de tipo booleano), procesar casos (de tipo booleano) y fichero donde el usuario ha definido las características específicas del dominio (de tipo File).
Tipo de tarea: resolución.

Al igual que en la anterior tarea, en esta capa también es necesario el uso de un fichero de texto externo donde el usuario de la aplicación ha definido una serie de expresiones con características ("features") específicas del dominio. El formato de estas reglas es el siguiente:

[FeatureName]{FeaturePosition}FeatureRegularExpresion

- FeatureName: es el nombre del atributo de la estructura del caso donde se ha de almacenar la información extraída.
- FeaturePosition: indica la posición que ocupa la información que queremos extraer dentro de la expresión regular definida a continuación. Esta posición viene definida por un número, que indica el número de paréntesis abiertos que hay que contar de izquierda a derecha para encontrar la información deseada.
- FeatureRegularExpression: expresión definida siguiendo la sintaxis que se especifica en la clase Java java.util.regex.Pattern.

Ejemplo: [Company]{1}((\p{Lu}\w+\s)+)(Inc\.|Corporation|Associates|Bank)

El ejemplo anterior se interpreta de la siguiente forma: si en el texto introducido por el usuario se encuentra la expresión definida en la tercera parte de la regla, en el atributo "Company" de la query se almacenará la información definida en la posición 1 de la expresión ((\p{Lu}\w+\s)+), que en este caso se corresponde con el nombre de la empresa.

Por ejemplo, el fragmento "Monsters Inc." coincide con la estructura definida en la expresión anterior, por lo que "Monsters" (correspondiente al grupo 1 de la expresión) será la información que se almacenará en el atributo "Company" de la query.

Al igual que en la etapa anterior, la información extraída del texto en esta capa se almacenará de forma provisional como parte del texto, para después ser copiada durante la etapa de extracción de información.

Para el dominio concreto de los viajes, las reglas que hemos introducido son las siguientes:

```
[HolidayType]{1}(Bathing|Active|City|Recreation|Wandering|Language|Education
|Skiing)

[Price]{3}(\p{Sc}(\s+))((\d+)([.,]\d+)?)
[Price]{1}((\d+)([.,]\d+)?)((\s+)\p{Sc})
[Price]{3}(([Ee]uro(s?)|[Dd]ollar)(s?)(\s+))((\d+)([.,]\d+)?)
[Price]{1}((\d+)([.,]\d+)?)((\s+)([Ee]uro(s?)|[Dd]ollar(s?)))
[Price]{1}((\d+)([.,]\d+)?)((\s+)(or|and|to)(\s+)((\d+)([.,]\d+)?)(\s?)(\p{Sc}
|[Ee]uro(s?)|[Dd]ollar(s?)))

[NumberOfPersons]{1}(\d+)((\s+)(person|persons|people))
[NumberOfPersons]{1}(\d+)((\s+)(or|and|to)(\s+)(\d+)(\s+)(person|persons|people))

[Region]{1}(Egypt|Cairo|Belgium|Bulgaria|Bornholm|Fano|Lolland|Allgaeu|Alps|Bavaria
|ErzGebirge|Harz|NorthSea|BalticSea|BlackForest|Thuringia|Atlantic|CotedAzur
|Corsica|Normandy|Brittany|Attica|Chalkidiki|Corfu|Crete|Rhodes|England|Ireland
|Scotland|Wales|Holland|AdriaticSea|LakeGarda|Riviera|Tyrol|Malta|Carinthia
|SalzbergerLand|Styria|Algarve|Madeira|Sweden|CostaBlanca|CostaBrava|Fuerteventura
|GranCanaria|Ibiza|Mallorca|Teneriffe|GiantMountains|TurkishAegeanSea
|TurkishRiviera|Tunisia|Balaton|Denmark|Poland|Slowakei|Czechia|France)

[Transportation]{1}(Plane|Car|Train|Coach)

[Duration]{1}(\d+)(\s+)(day|days)
[Duration]{1}(\d+)((\s+)(or|and|to)(\s+)(\d+)(\s+)(day|days))

[Season]{1}(January|February|March|April|May|June|July|August|September|October
|November|December)

[Accommodation]{1}(OneStar|TwoStars|ThreeStars|HolidayFlat|FourStars|FiveStars)
```

Para los atributos cuyo tipo es una enumeración de valores (tipos enumerados), las reglas de esta capa lo único de que se encargan es de identificar éstos dentro del texto del usuario y almacenarlos en el atributo correspondiente.

A continuación se explica un pequeño ejemplo para ilustrar el trabajo realizado en esta etapa. Si el usuario introduce como consulta el texto "I would like to travel to Fuerteventura by Plane or Car", en esta capa de extracción de información se detectarán dos características específicas del dominio:

- **Fuerteventura.** La única regla existente para el atributo "Region" detectará el nombre del destino del viaje y almacenará la información asociándola al texto de consulta.

- **Plane.** Al igual que en el caso anterior, la regla correspondiente detectará el medio de transporte introducido por el usuario y almacenará esa información asociada al atributo "Transportation".
- **Car.** Al igual que en el caso anterior, la misma regla detectará el valor "Car" y asociará la información pertinente al texto de consulta.

Cuando alguna de las reglas anteriores detecte en la consulta del usuario algún fragmento reconocido, se asociará un nuevo valor a la lista de valores del atributo "Features" del párrafo del texto donde se haya encontrado. Este nuevo valor se encapsulará en un objeto de tipo "FeatureInfo" con los siguientes datos:

- Nombre del atributo de la estructura de casos especificado en la regla.
- Valor reconocido en el texto de consulta del usuario.
- Posiciones de comienzo y final del fragmento del texto reconocido.

En el ejemplo anterior, al sólo estar compuesto el texto de un párrafo se almacenará en su lista de valores de su atributo "Features" los siguientes objetos "FeatureInfo":

- FeatureInfo1 = { Region, Fuerteventura, 27, 39 }
- FeatureInfo2 = { Transportation, Plane, 44, 48 }
- FeatureInfo3 = { Transportation, Car, 53, 55 }

Capa de estructura de dominio (Domain Structure Layer)

Localización: CBR Cycle → Obtain Query Task → Feature Value Layer.
Clase Java que implementa la tarea: jcolibri.extensions.textual.method.DomainTopicClassifierMethod
Parámetros: procesar query (de tipo booleano), procesar casos (de tipo booleano) y fichero donde el usuario ha definido las estructuras características del dominio (de tipo File).
Tipo de tarea: resolución.

Al igual que las tareas anteriores, esta etapa requiere el uso de un fichero externo donde el usuario de la aplicación ha definido una serie de reglas con estructuras características del dominio en el que se trabaja, también llamadas "temas" ("topics"). En este caso, la estructura de este tipo de reglas es el siguiente:

[Topic] <FeatureName,value> <FeatureName,value> ... <Phrase> <Phrase>

- Topic: nombre con el que se clasifica el tema.
- FeatureName: referente al FeatureName definido en la capa "Feature Value Layer".
- Value: valor del FeatureName anterior. Puede ser ?, indicando cualquier valor posible.
- Phrase: posibilidad de utilizar las expresiones definidas en la capa "Phrase Layer".

Ejemplo: [ProjectX]<University, Oxford><Person, John><Company,?><Compaq Presario 2100>

Este tipo de reglas sirven para realizar una clasificación por temas. En el caso del ejemplo, si en las capas anteriores se ha determinado el valor "Oxford" para el atributo "University", el valor "John" para el atributo "Person", cualquier valor para el atributo "Company" y se ha detectado la expresión "Compaq Presario 2100", entonces el sistema sabrá que estamos hablando del proyecto "ProjectW", y hará la clasificación de tal forma.

En el dominio específico de los viajes, este tipo de reglas no son utilizadas.

Capa de extracción de información (Information Extraction Layer)

Localización: CBR Cycle → Obtain Query Task → Information Extraction Layer.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.TextualQueryIEMethod.
Parámetros: fichero XML donde se ha definido la estructura de los casos del sistema (de tipo File).
Tipo de tarea: resolución.

En esta tarea el sistema debe almacenar toda la información extraída en las capas anteriores en una nueva query con una estructura de atributos idéntica a la de todos los casos de la base de casos. Como se ha comentado en la descripción de las etapas anteriores, las expresiones, características y temas encontrados se han almacenado de forma provisional como parte del texto del usuario. Es en esta capa en la que se extraerá esta información y se almacenará en los atributos correspondientes.

En la implementación de esta etapa se utilizan los métodos de extracción de información que proporciona el entorno jCOLIBRI en la clase jcolibri.extensions.textual.method.BasicIEMethod:

- **ExtractTopicInfo.** Método que almacena la información extraída en la capa "Domain Structure Layer" en los atributos correspondientes de la nueva query.
- **ExtractFeaturesInfo.** Método que almacena la información extraída en la capa "Feature Value Layer" en los atributos correspondientes de la nueva query.
- **ExtractPhrasesInfo.** Método que almacena la información extraída en la capa "Phrase Layer" en los atributos correspondientes de la nueva query.

La primera tarea de esta etapa del sistema es construir una nueva query que, en vez de un solo parámetro de tipo texto como la primera que se creó, tenga la misma estructura que los casos contenidos en la base de casos. Para ello se le pasará como referencia un fichero XML donde está definida esta estructura. Estos nuevos atributos se inicializarán a un valor nulo.

Después, se realizarán llamadas a los métodos de extracción de información anteriores, con parámetros el texto de la consulta del usuario y la nueva query estructurada. Estos métodos buscarán las expresiones encontradas en las capas anteriores y almacenarán esa información en los atributos correspondientes de la nueva query, cuya referencia le estamos pasando.
Lo último que resta por hacer en esta capa es guardar la nueva query estructurada en el contexto del sistema, de forma que las capas que están por venir puedan acceder a ella. A partir de este momento, el texto de consulta escrito por el usuario al comienzo de la ejecución se perderá, y en su lugar se almacenará una nueva query con la información ya extraída y lista para ser comparada en similitud con la base de casos.

Para ilustrar esta etapa retomaremos nuestro ejemplo anterior ("I would like to travel to Fuerteventura by Plane or Car"). Como ya vimos antes, en la capa "Feature Value Layer" se encontraron tres "features" y se asoció la información necesaria al único párrafo que conforma el texto. Por lo tanto, en esta etapa, el método "ExtractFeaturesInfo" es el encargado de recuperar esta información y almacenarla en los atributos correspondientes de la nueva query estructurada. Por cada objeto "FeatureInfo" asociado encontrado este método se encarga de buscar entre los nuevos atributos si alguno coincide en nombre con el especificado en la regla de la etapa anterior. En nuestro ejemplo, el método buscaría los atributos con nombre "Region" y "Transportation", y los encontraría. El siguiente paso entonces sería almacenar el valor asociado y que se corresponde con el encontrado en la consulta del usuario. Por tanto:

Query
Region = Fuerteventura
Transportation = Plane Car

En el caso del atributo "Transportation", como se han encontrado dos valores posibles, ambos serán almacenados, dejando entre ambos un espacio en blanco. Si se hubiesen encontrado tres valores en vez de dos, se procedería de igual modo, siempre dejando un espacio en blanco entre dos valores.

Como se puede observar, en cada atributo de la nueva query se van almacenando los valores encontrado por las capas anteriores, siempre en forma de cadena de texto. Es obligación de las siguientes etapas el transformar estos datos a otros tipos (enteros, tipos enumerados...) si así lo requieren.

Como ya se ha detallado antes, si para un atributo de la query no se han encontrado valores en la consulta del usuario, éste tomará un valor nulo.

Tarea Recuperar (Retrieve Task)

Localización: CBR Cycle → Retrieve Task.
Clase Java que implementa la tarea: jcolibri.method.RetrieveComputationalMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

Esta tarea se encarga de realizar la recuperación de casos del sistema CBR. Para ello, lo primero que hace es seleccionar los casos de la base de casos que entran a formar parte de esta recuperación. Después, ejecuta las funciones de similitud, tanto globales como locales, entre la query del usuario y el resto de casos de la base. Por último, recupera el caso o casos (según se especifique) con mayor similitud con las preferencias del usuario, según ha indicado mediante su consulta textual.

Tarea de selección de casos activos (Select working cases task)

Localización: CBR Cycle → Retrieve Task → Select working cases task.
Clase Java que implementa la tarea: jcolibri.method.SelectAllCasesMehtod.
Parámetros: ninguno.
Tipo de tarea: resolución.

Esta tarea se encarga de seleccionar los casos que entran a formar parte en el proceso de recuperación del sistema CBR. Para el dominio específico de los viajes (y para casi todos los demás), se seleccionan todos los casos obtenidos en la primera etapa del sistema.

Por tanto, lo único que realiza este método es almacenar todos los casos de la base en el contexto del sistema, para que las etapas que vienen a continuación puedan utilizarlos.

Tarea de cómputo de similitud (Compute similarity task)

Localización: CBR Cycle → Retrieve Task → Compute similarity task.
Clase Java que implementa la tarea: jcolibri.method.NumericSimComputationMethod.
Parámetros: ninguno.
Tipo de tarea: resolución.

Esta tarea es la encargada de obtener la similitud numérica entre la query del usuario y cada uno de los casos contenidos en la base de casos, mediante la aplicación de las funciones de similitud locales y globales. Estas funciones de similitud se definen en ficheros XML donde se especifica la estructura del caso, con sus atributos, sus tipos y sus parámetros.

Las funciones de similitud locales son las que se aplican para calcular la similitud entre dos valores diferentes de un mismo atributo, entre la query y un caso cualquiera. Las funciones de similitud globales son las que calculan numéricamente cuánto se parece la query del usuario a cada caso, mediante el cómputo de los valores obtenidos por las funciones de similitud locales para cada atributo. En ambos

casos, el valor devuelto por estas funciones se encuentra en un intervalo entre 0 y 1: cuanto mayor sea el número, más se parecen los dos valores comparados.

Para el dominio específico de los viajes, hemos especificado las siguientes funciones de similitud:

- **HolidayType**
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = HolidayTypesEnum

- **Price**
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 2500

- **NumberOfPersons**
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 12

- **Region**
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = RegionsEnum

- **Transportation**
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = TransportationsEnum

- **Duration**
  Función de similitud local: AverageMultipleTextValues
  Parámetro de la función de similitud: Interval = 21

- **Season**
  Función de similitud local: MinEnumCyclicDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = SeasonsEnum
- **Accommodation**
  Función de similitud local: MinEnumDistanceMultipleTextValues
  Parámetro de la función de similitud: Enumeration = AccomodationsEnum

- **Hotel**
  Función de similitud local: EqualsStringIgnoreCase
  La función de similitud especificada no requiere el uso de parámetros.

  Función de similitud global: **Average**.

A continuación se explica en detalle el comportamiento de cada una de las funciones de similitud anteriores:

- **Average.** Esta función global calcula el valor numérico de similitud entre la query del sistema y un caso específico mediante el cálculo de la media aritmética entre los valores resultantes de aplicar las funciones de similitud locales entre todos los atributos de ambos.

- **AverageMultipleTextValues.** Esta función local convierte valores de atributos de tipo cadena de texto (String) en un conjunto de valores numéricos y calcula su media. Después, calcula la diferencia dentro de un intervalo (especificado como parámetro de la función) entre la query del usuario y el caso correspondiente.

Ejemplo:

Query	10 18	MediaQ = (10 + 18) / 2 = 14
Caso	10 12 14	MediaC = (10 + 12 + 14) / 3 = 12
Intervalo	20	
Resultado	1 - (| MediaQ - MediaC | / Intervalo) = 0.9	

- **MinEnumDistanceMultipleTextValues.** Esta función local extrae de una cadena de texto los tokens que la forman y calcula el ordinal de cada uno de ellos dentro del tipo enumerado que recibe como parámetro (posición de ese valor dentro de la enumeración), tanto para la query como para el caso que se está evaluando. El valor devuelto por la función será la mínima distancia encontrada entre los ordinales de la query y los del caso, ponderada por el número total de valores del tipo enumerado.

  Ejemplo:

  Tipo Enumerado: Plane (1), Car (2), Train (3), Coach (4)

Query	Plane Car	Ordinales Q = {1, 2}
Caso	Coach	Ordinales C = {4}
Distancias	|1 - 4| = 3 , |2 - 4| = 2	→ {2, 3}
Mínima distancia	2	
Resultado	1 - (Min. Dist. / Num. Valores) = 0.5	

- **MinEnumCyclicDistanceMultipleTextValues.** Esta función local se comporta de igual forma que la función anterior, con la diferencia que la diferencia entre ordinales que se calcula es cíclica, es decir, la distancia entre el primero y el último elemento de la enumeración es 1, mientras que para la función anterior la diferencia es el número de posibles valores menos 1.

  Ejemplo:

  Tipo Enumerado: Plane (1), Car (2), Train (3), Coach (4)

Query	Plane Car	Ordinales Q = {1, 2}
Caso	Coach	Ordinales C = {4}
Distancias no cíclicas	|1 - 4| = 3 , |2 - 4| = 2	→ {2, 3}
Distancias cíclicas	|4 - 3| = 1 , |4 - 2| = 2	→ {1, 2}
Mínima distancia	1	
Resultado	1 - (Min. Dist. / Num. Valores) = 0.75	

- **EqualsStringIgnoreCase.** Esta función de similitud local compara dos cadenas de texto sin tener en cuenta la capitalización de sus caracteres (si son letras mayúsculas o minúsculas). Si ambas cadenas son iguales el resultado devuelto será 1 y en caso contrario 0.

  Ejemplo:

Query	hotel ritz
Caso	Hotel Ritz
Resultado	1

En cualquiera de las funciones anteriores, si los parámetros a comparar no son del tipo esperado o son nulos, el resultado retornado por ésta será de 0, lo que significa que ambos no tienen ninguna similitud.

Tarea de selección del mejor (Select best task)

Localización: CBR Cycle → Retrieve Task → Select best task.
Clase Java que implementa la tarea: jcolibri.method.SelectBestCaseMethod.
Parámetros: ninguno.
Tipo de tarea: resolución.

Esta tarea se encarga de seleccionar el caso de la base de casos con una mayor similitud con la query que representa la consulta del usuario, utilizando para ello los valores calculados en la etapa anterior.

Tareas Reutilizar, Revisar y Recordar (Reuse, Revise and Retain taks)

En el dominio específico de los viajes en el que nosotros estamos trabajando no es necesario el uso de estas tres etapas, ya que lo único que nos interesa

Post-ciclo (PostCycle)

Localización: PostCycle (etapa principal).
Clase Java que implementa la tarea: jcolibri.method.CloseConnectorMethod.
Parámetros: ninguno.
Tipo de tarea: resolución.

El post-ciclo de este sistema CBR sólo se encarga de cerrar el conector que se abrió en el pre-ciclo del mismo para obtener los casos de la base de datos. Ésta es la última etapa del sistema.

Nuevas utilidades añadidas

En esta primera aproximación de nuestro proyecto no han sido muchas las utilidades que se han añadido al entorno de jCOLIBRI. Más bien, esta primera parte ha servido como toma de contacto y base para comenzar a desarrollar nuestro sistema de asistencia.

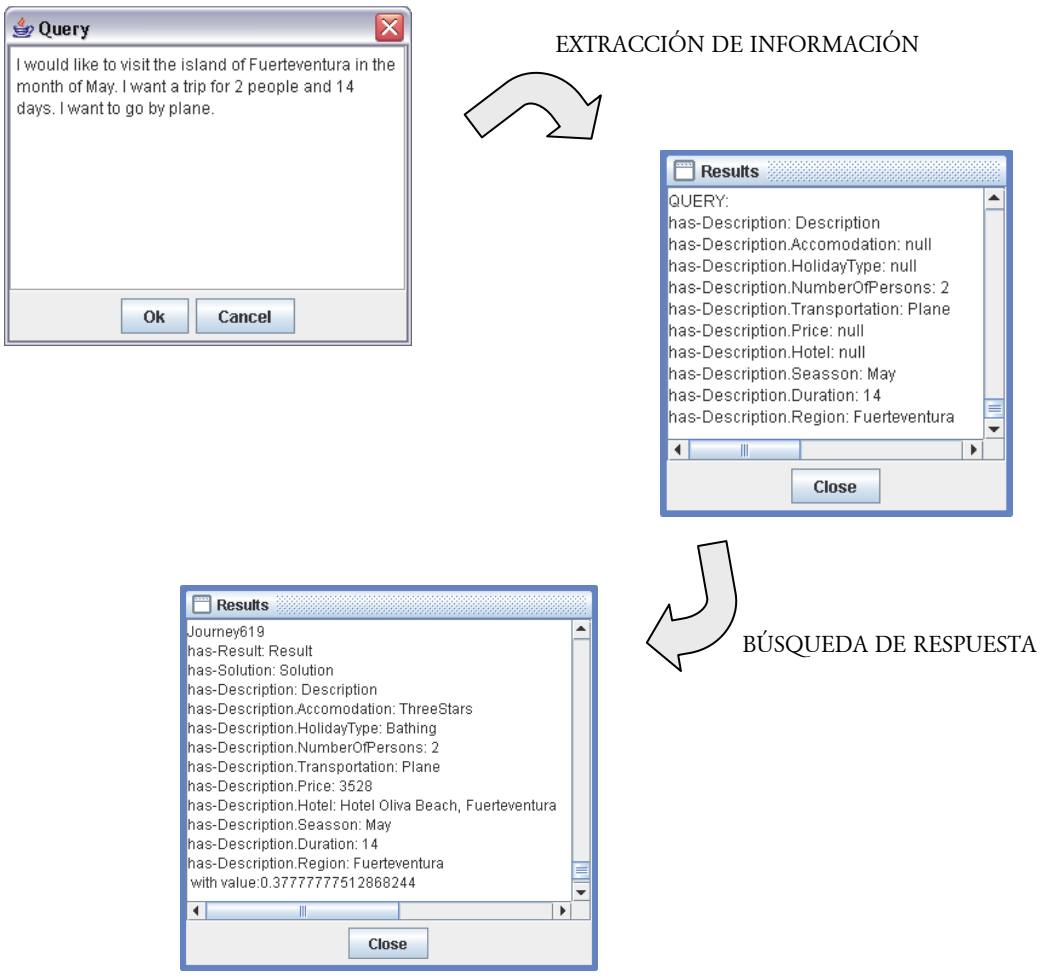
Las nuevas funcionalidades añadidas en esta primera etapa han sido:

- Permitir consultas sólo textuales.
- Nuevo conocimiento para el dominio de viajes.
- Tratamiento de las consultas textuales.
- Nuevas funciones de similitud local.

Permitir consultas sólo textuales

En uno de los ejemplos de CBR Textual proporcionado con el entorno jCOLIBRI, el tipo de consultas que puede realizar el usuario son "mixtas", es decir, puede describir su consulta de forma textual pero también puede fijar valores específicos para cada uno de los atributos que conformar la estructura de casos del sistema.

Con el trabajo que hemos realizado en esta aproximación, el sistema sólo permite consultas textuales, a partir de las cuales se extraerá la información relevante con la que fijar los valores de los atributos.



Nuevo conocimiento para el dominio de viajes

Según se ha explicado en este apartado, en algunas de las etapas del sistema CBR es necesaria la utilización de ficheros externos en los que se han definido una serie de reglas para la extracción de información a partir de textos de consulta. Estas reglas contienen el "conocimiento" sobre el dominio, ya que en ellas se especifican expresiones, construcciones o características típicas de éste.

Las reglas utilizadas en este primer sistema CBR han sido creadas por nosotros, de forma que contienen el máximo conocimiento posible sobre el dominio, aunque siempre son susceptibles de ser mejoradas con una mayor cantidad de información.

Tratamiento de las consultas textuales

Como se ha comentado en el primer punto de este apartado, nuestro sistema sólo permite consultas textuales, de forma que los usuarios escriben sus descripciones y éste extrae la información relevante. Para que el tratamiento de dicha información se realice de la manera más óptima, nos hemos adaptado lo máximo posible al código ya escrito en el entorno jCOLIBRI, por lo que sólo ha sido necesario realizar unos mínimos cambios.

Por ejemplo, para almacenar la consulta textual del usuario hemos creado un nuevo tipo de query (CBRTextualQuery) que hereda todas las propiedades de las querys genéricas ya diseñadas (CBRGenericQuery). Esta primera "query textual" es almacenada dentro del contexto del sistema (CBRContext), que va pasando de capa en capa, como cualquier otra, sin necesidad de utilizar un contenedor diferente, y aprovechando el que nos proporciona esta estructura de comunicación. Cuando se llega a la etapa de extracción de información, esta query textual se sustituye por una query estructurada en el contexto del sistema. De esta forma, no es necesario diferenciarlas.

Otro ejemplo de este comportamiento es la utilización de los métodos de extracción de información "ExtractTopicInfo", "ExtractFeaturesInfo" y "ExtractPhrasesInfo" en el método desarrollado por nosotros "TextualQueryIEMethod".

Nuevas funciones de similitud local

En esta primera aproximación se han diseñado dos nuevas funciones de similitud para la comparación de valores sobre tipos enumerados:

- MinEnumDistanceMultipleTextValues
- MinEnumCyclicDistanceMultipleTextValues

La realización de estas dos nuevas funciones ha sido necesaria debido al cambio de tipos en los atributos de la estructura de casos que se ha tenido que realizar. Estos atributos han dejado de ser tipos enumerados para convertirse en cadenas de texto, ya que la información almacenada en ellos procede de fragmentos de la consulta del usuario. Por tanto, las dos funciones anteriores comparan dos cadenas de texto, una de la query y otra de un caso de la base de casos, dividiendo ambas en tokens que, supuestamente, contienen valores de un cierto tipo enumerado, que es pasado como parámetro a la función para que realice el cálculo de similitud.

Limitaciones del sistema

Aunque con el trabajo realizado en esta primera aproximación se ha dado un gran paso para tratar de forma correcta las consultas textuales, existen algunas limitaciones en el comportamiento y la flexibilidad de la aplicación. A continuación se enumeran algunas de esas limitaciones:

- Conocimiento contenido en las reglas provistas por el usuario.
- Limitación en los valores de los tipos enumerados.
- El sistema no permite la representación de negaciones, conjunciones y disyunciones.
- El sistema no permite representar atributos con valores múltiples.
- El usuario no puede confirmar sus datos.
- El sistema no es capaz de obtener información relativa a un determinado concepto.

Conocimiento contenido en las reglas provistas por el usuario

Como ya hemos visto en este apartado, en esta aproximación el conocimiento específico del dominio (en este caso, el de los viajes) se incluye mediante el diseño de una serie de reglas. Estas reglas son definidas por el usuario del sistema y constituyen una sencilla forma de incluir palabras, expresiones y características específicas del dominio con el que se trabaja.

Pero el principal problema de esta aproximación aparece cuando el usuario no proporciona este conocimiento propio sobre el dominio. Así, en las correspondientes etapas del sistema CBR dónde se utilizan las reglas no realizarán tarea alguna, por lo que el análisis y posterior extracción de información sobre las consultas textuales no tendrá ningún resultado. De esta forma, no será posible realizar un trabajo acorde a las necesidades, lo que conllevará que no se podrá responder al usuario con unos datos adecuados a sus preferencias.

Cuanto mayor sea la precisión y la cantidad de información almacenada en las reglas definidas por el usuario de la herramienta, mejor será el resultado esperado. El problema viene cuando estos datos son incompletos o, incluso, no existen.

Limitación en los valores de los tipos enumerados

Algunos de los tipos de los atributos de la estructura de casos del dominio específico de los viajes son enumerados, es decir, están formados por una serie de valores posibles. Estos valores son cadenas de texto que representan una serie de conceptos propios del dominio.

Uno de los ejemplos de estos tipos enumerados en nuestro dominio de los viajes es el que representa las categorías de los alojamientos, cuyo nombre es "AccommodationsEnum" y está compuesto por los siguientes valores:

- OneStar
- TwoStars
- ThreeStars
- HolidayFlat
- FourStars
- FiveStars

Como se puede observar, cada uno de los valores de esta enumeración representa un concepto dentro del dominio de los viajes y, concretamente, en las categorías de los alojamientos (en este caso, el número de estrellas de los hoteles).

Estos valores también han sido incluidos como parte del conocimiento en forma de reglas (ver descripción del sistema CBR en este apartado), de forma que puedan ser reconocidos mediante el análisis del texto de consulta si los usuarios se refieren a ellos.

Pero, como es de esperar, existe una remota posibilidad de que estos valores sean escritos por los usuarios tal cual, es decir, con la misma expresión (OneStar, TwoStars...). Aquí es donde radica la principal limitación en el uso de estos conceptos. En la segunda aproximación del proyecto se ofrece una sencilla solución a este problema, de forma que el usuario de la herramienta pueda definir expresiones con el mismo significado que estos conceptos y que permitan ser reconocidos sin necesidad de escribir las expresiones originales.

El sistema no permite la representación de negaciones, conjunciones y disyunciones

En la capa de extracción de información del sistema CBR se ha visto que los valores almacenados en la query estructurada están representados mediante una enumeración, con espacios en blanco entre cada dos. Ésta es una forma "plana" de representar la información obtenida de las consultas de texto de los usuarios y no permite apenas flexibilidad. Para un sistema sencillo, ésto no supone ningún problema, pero en un trabajo más elaborado podemos darnos cuenta de que esta aproximación no nos permite representar apenas información.

Un ejemplo claro de esta información no representable son las negaciones, conjunciones y disyunciones, que son utilizadas en infinidad de expresiones del lenguaje natural por las personas, y que necesitan ser representadas si no queremos obviar información relevante. De momento, el entorno jCOLIBRI no tiene desarrollada una infraestructura que permita representar estos datos.

En esta primera aproximación de nuestro proyecto se han obviado estas conjunciones, y las enumeraciones de valores son consideradas como disyunciones (como oes). A continuación se expone un pequeño ejemplo que ilustra este tipo de situaciones:

Query 1	"I would like to go to Mallorca **and** Fuerteventura"
Query 2	"I would like to go to Mallorca **or** Fuerteventura"
Query 3	"I do **not** want to go to Mallorca or Fuerteventura"

Información extraída en los tres casos: Region = "Mallorca Fuerteventura"

Como se puede observar, las tres consultas anteriores tienen significados diferentes, pero el sistema extrae idéntica información en los tres casos, obviando de esta forma las conjunciones, disyunciones y negaciones necesarias para dotar a la consulta de una semántica correcta.

El sistema no permite representar atributos con valores múltiples

Como ya se ha visto en el punto anterior, la información almacenada por el sistema en los atributos de la query estructurada es siempre una cadena de texto que contiene una serie de valores separados dos a dos mediante espacios en blanco. Si deseamos trabajar con tipos múltiples (múltiples valores para un mismo atributo), esta representación obliga a que todos los atributos sean de tipo cadena de texto (String), siendo las funciones de similitud las encargadas de transformar estos datos para poder trabajar con ellos. Es decir, si se trata de una sucesión de enteros, la función deberá de extraer cada uno de estos valores de la cadena de texto y trabajar con ellos según lo requiera.

jCOLIBRI no proporciona tipos múltiples, por lo que sólo se puede utilizar la representación anterior. Para la tercera aproximación de nuestra proyecto se han definido una serie de tipos múltiples (enteros múltiples y tipos enumerados múltiple) que permiten la representación deseada de una forma sencilla.

El usuario no puede confirmar sus datos

En esta aproximación el sistema CBR acepta una consulta textual de un usuario, la analiza, extrae la información que considera relevante y devuelve una respuesta. Si alguna parte del sistema funciona de forma incorrecta, probablemente la respuesta también sea incorrecta, o no se ajuste a las preferencias del usuario. Por tanto, una buena utilidad sería añadir una pequeña infraestructura que permita al usuario verificar si la información extraída es correcta, pudiendo modificar dichos datos.

Para la tercera aproximación del proyecto se ha realizado el diseño de esta utilidad que la versión básica de jCOLIBRI no dispone.

El sistema no es capaz de obtener información relativa a un determinado concepto

Es posible que nuestro sistema reconozca y extraiga de las consultas textuales de los usuarios datos y conceptos que no coincidan con ninguno de los valores contenidos en la base de conocimiento del sistema CBR. De esta forma, al aplicar las funciones de similitud no existirá coincidencia y se devolverá un valor de 0 para el atributo correspondiente que contenga dichos datos. En el caso de que esta situación suceda para todos los valores extraídos de la consulta del usuario, la similitud con todo la base de casos del sistema será nula, por lo que no se podrá devolver ningún caso, ya que todos tendrán coincidencia 0. Así, nuestro sistema no podrá generar una respuesta para el usuario. Éste es el comportamiento actual.

Un sistema más elaborado intentaría buscar información relativa en el caso de que no obtuviera resultados tras la consulta del usuario. De esta forma, para un determinado concepto se extraerían otros que tuvieran relación con el primero, para poder responder al usuario con una recomendación similar. Por ejemplo, consideremos que el usuario escribe en su consulta textual que desea viajar a la isla de Fuerteventura, pero lamentablemente este destino no está disponible entre los que se ofertan, lo que es equivalente a decir que este valor no se encuentra en la base de conocimiento. Nuestro sistema actual sería incapaz de encontrar similitudes, por lo que no podría devolver respuesta alguna. Con la nueva utilidad planteada, el sistema buscaría información relativa al concepto "Fuerteventura", es decir, destinos similares que también pueden satisfacer las exigencias del usuario. De esta forma, el sistema podría ofrecerle otros destinos como Tenerife, Gran Canaria o Lanzarote.

En la cuarta aproximación del proyecto se desarrollará una nueva utilidad para nuestro sistema que encuentre información relativa para un determinado concepto, mediante la utilización de ontologías.

Problemas encontrados

En esta primera aproximación de nuestro proyecto hemos encontrado algunos problemas con los que hemos tenido que convivir y resolver. Estos son algunos de ellos:

- Adaptación al entorno de jCOLIBRI.
- Cambio de los tipos de los atributos de la estructura de casos.
- Tratamiento de las consultas textuales.
- Dudas con el objetivo de la aproximación.

Adaptación al entorno de jCOLIBRI

La primera gran dificultad que nos encontramos a la hora de comenzar con esta primer aproximación de nuestro proyecto a la herramienta que íbamos a utilizar. El manejo y funcionamiento de jCOLIBRI no tiene mucha dificultad, pero siempre es difícil adaptarse a una nueva aplicación.

Para empezar el desarrollo de nuestro trabajo, estudiamos a fondo todas las características de jCOLIBRI: tareas, método, estructuras de casos, conectores, funciones de similitud, tipos..., de forma que así consiguiéramos familiaridad con todo el entorno.

En los comienzos de nuestro trabajo muchas veces nos preguntábamos si ciertas cosas que queríamos hacer era posible llevarlas a cabo con esta herramienta. Con el tiempo nos dimos cuenta que sólo era cuestión de aprendizaje e insistencia. Muchas veces también se nos plantearon problemas a la hora de añadir nuestras nuevas funcionalidades a las ya disponibles en la aplicación.

Cambio de los tipos de los atributos de la estructura de casos

Como ya hemos explicado a lo largo de este apartado, todos los tipos de los atributos de la estructura de casos son de tipo cadena de texto (String). Ésto es así debido a que los valores que contienen proceden de fragmentos del texto de consulta de los usuarios de la aplicación. El significado de estos valores es el mismo independientemente del tipo que tengan sus atributos contenedor, lo único que cambia es la representación de éstos. Por tanto, son las funciones de similitud las encargadas de transformar estos datos y adaptarlos para poder trabajar con ellos.

Antes de llegar a esta serie de conclusiones se nos planteó un problema de incompatibilidad entre los tipos de los atributos de la estructura de casos y los tipos de los valores que éstos contienen. La solución no fue trivial: si cambiábamos todos los tipos de los atributos a cadenas de texto se perdía mucha información relevante, por ejemplo, los tipos enumerados; y si manteníamos los tipos tal cual se producía una incompatibilidad con la información que era extraída de los textos de consulta de los usuarios.

Finalmente, se tomó la alternativa de asignar todos los tipos como cadenas de texto, para poder almacenar sin problemas la información relevante extraída, pero con algunas condiciones. De esta forma, las funciones de similitud de cada atributo son las que se encargan de tratar los datos de forma adecuada y según sus necesidades. Así, y como ya se ha visto antes, existen funciones que tratan cadenas de texto como una correlación de números, u otras que consideran estos datos como valores dentro de una enumeración de elementos. Para este tipo específico de funciones en las que intervienen tipos enumerados ha sido necesario proporcionar a estas funciones las enumeraciones de valores que han de utilizar.

Tratamiento de las consultas textuales

Éste fue uno de los principales problemas que nos encontramos en esta primera etapa de proyecto. Según se ha explicado, la extensión textual que proporciona el entorno jCOLIBRI permite consultas textuales y con selección de parámetros, mientras que nuestro objetivo era permiten sólo el primer tipo. Es por ésto que en un primer momento no tuvimos claro como diseñar nuestra implementación. Con el consecuente estudio de posibilidades conseguimos ingeniar la forma de tratar este tipo de comportamiento.

Dudas con el objetivo de la aproximación

Por el mismo motivo que en el punto anterior, hubo momentos durante el desarrollo de nuestro trabajo en esta primera aproximación en los que no tuvimos muy claro cuál era nuestro principal objetivo a conseguir. Tras unas pequeñas redefiniciones sobre éste, el trabajo continuó de forma regular.

Pruebas realizadas

En es último apartado de esta primera fase se detallan algunas pruebas que se han llevado a cabo para ver cuál es el funcionamiento de nuestro sistema. Para ello, se han redactado algunas consultas de ejemplo que han sido ejecutadas en el sistema para comprobar qué información relevante es capaz de extraer.

En cada una de las consultas expuestas a continuación se han subrayado los datos importantes que deberían ser extraídos por el sistema, para facilitar su identificación.

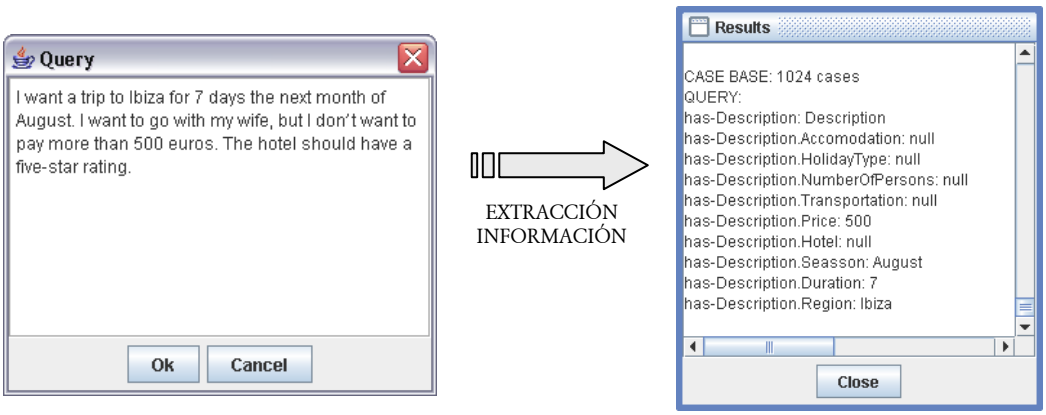
Prueba 1

Consulta

- I want a trip to Ibiza for 7 days the next month of August. I want to go with my wife, but I don't want to pay more than 500 euros. The hotel should have a five-star rating.

Información extraída

- Price = 500
- Season = August
- Duration = 7
- Region = Ibiza



En esta primera prueba el sistema es capaz de reconocer sólo algunos de los datos relevantes que contiene la consulta textual. En primer lugar, la etapa "Feature Value Layer" aplica las reglas que sido definidas por el desarrollador, con el objetivo de encontrar expresiones características y propias del dominio. Es en esta capa donde se reconocen y extraen los datos mostrados en la imagen para los atributos "Price", "Season", "Duration" y "Region". Estas son las reglas, definidas en el fichero de configuración de esta etapa, que permiten el reconocimiento de dicha información:

```
[Price]{1}((\d+)([.,]\d+)?)((\s+)([Ee]uro(s?)|[Dd]ollar(s?)))
```

```
[Seasson]{1}(January|February|March|April|May|June|July|August|September|October
|November|December)
```

```
[Duration]{1}(\d+)((\s*)(day|days))
```

```
[Region]{1}(Egypt|Cairo|Belgium|Bulgaria|Bornholm|Fano|Lolland|Allgaeu|Alps|Bavaria
|ErzGebirge|Harz|NorthSea|BalticSea|BlackForest|Thuringia|Atlantic|CotedAzur
|Corsica|Normandy|Brittany|Attica|Chalkidiki|Corfu|Crete|Rhodes|England|Ireland
```

```
|Scotland|Wales|Holland|AdriaticSea|LakeGarda|Riviera|Tyrol|Malta|Carinthia
|SalzbergerLand|Styria|Algarve|Madeira|Sweden|CostaBlanca|CostaBrava|Fuerteventura
|GranCanaria|Ibiza|Mallorca|Teneriffe|GiantMountains|TurkishAegeanSea
|TurkishRiviera|Tunisia|Balaton|Denmark|Poland|Slowakei|Czechia|France)
```

Estas reglas permiten reconocer patrones en el texto de consulta del usuario y extraer la información encontrada para ser almacenada en los atributos correspondientes de la query del sistema CBR.

Como se puede comprar, existen una serie de datos en la consulta que son relevantes pero que no son exreaídos por nuestro sistema. Éstos son los motivos de dicho comportamiento para cada uno de esos conceptos:

- A partir del fragmento del texto de consulta "I want to go with my wife" ("Quiero ir con mi mujer") el sistema debería entender que el número de personas del viaje es 2, es decir, que el valor para el atributo "NumberOfPersons" de la query debe ser 2. Si una persona cualquiera lee esta descripción entiende que el que ha escrito la consulta intenta comunicar que quiere un viaje para él y para su mujer, es decir, que el total de personas es dos. En cambio, nuestro sistema no es capaz de reconocer esta expresión y, por tanto, no puede extraer dicha información. Éste es considerado uno de los principales problema, ya que existen una gran cantidad de maneras de especificar el número de pasajeros de un viaje, como por ejemplo mediante la enumeración de las personas que acuden ("Deseo viajar con mis dos hermanos y mi amigo Luis").

- Para el caso del fragmento "five-star rating", el usuario está especificando que desea que el hotel donde se aloje sea de cinco estrellas, es decir, que el valor para el atributo "Rating" ha de ser "FiveStars". En esta ocasión, el sistema no es capaz de extraer este dato, ya que la única forma que tiene de reconocerlo es si el usuario introduce la cadena de texto "FiveStars", lo cual es muy poco probable, ya que se trata más bien de una representación interna del concepto. En la siguiente fase de desarrollo de nuestro proyecto ya será posible reconocer este tipo de datos mediante el uso de construcciones propias del lenguaje o expresiones específicas del dominio.

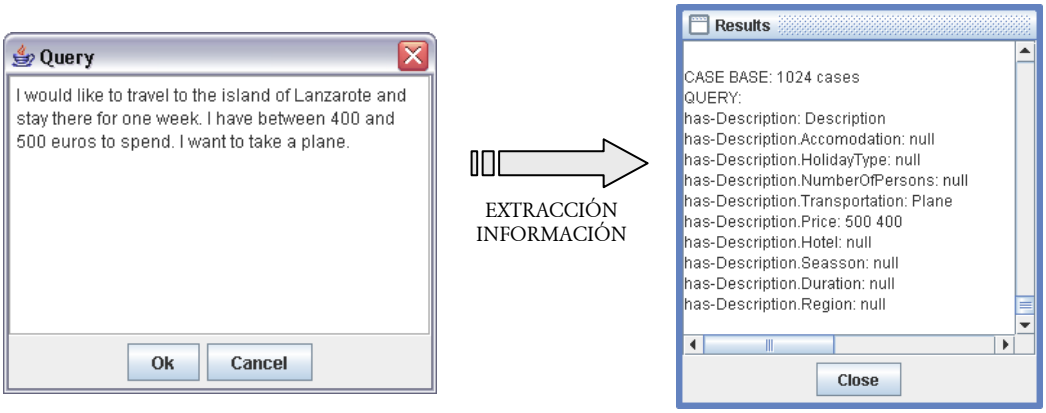
Prueba 2

Consulta

- I would like to travel to the island of Lanzarote and stay there for one week. I have between 400 and 500 euros to spend. I want to take a plane.

Información extraída

- Transportation = Plane
- Price = 500 400



En este segundo ejemplo de prueba el sistema utiliza la etapa "Feature Value Layer" para extraer los dos datos que se pueden ver en la captura anterior. Las reglas utilizadas en este caso son las siguientes:

```
[Transportation]{1}(Plane|Car|Train|Coach)
```

```
[Duration]{1}(\d+)((\s*)(or|and|to)(\s*)(\d+)(\s*)(day|days))
```

La primera de estas reglas sirve para reconocer en los textos de consulta los medios de transporte de los diferentes viajes. En este caso, lo único que se verifica es que la cadena que representa el concepto dentro del CBR ("Plane") coincide con el fragmento de la consulta del usuario ("plane"), sin tener en cuenta las mayúsculas o las minúsculas. La segunda regla definida es utilizada para detectar en el texto intervalos de números enteros que se refieran al precio del viaje. De esta forma, es posible extraer de la consulta los valores 400 y 500 para almacenarlos en el atributo "Price" de la query.

Al igual que para la prueba anterior, nuestro sistema no es capaz de reconocer y extraer de la consulta todos los datos relevantes que ésta contiene. Los principales motivos son los siguientes:

- En el caso del valor "Lanzarote", el sistema no lo detecta ya que no existe ninguna regla en el fichero de características que lo reconozca. Aunque nosotros creáramos un patrón que fuera capaz de extraer de la consulta este dato, sería un trabajo inútil, ya que éste no se encuentra en la base de conocimiento del CBR, es decir, ningún viaje de la base de casos tiene como destino la isla de Lanzarote. Es por ésto por lo que para la cuarta fase de desarrollo de nuestro proyecto se ha llevado a cabo la implementación de una etapa que permita, mediante la interacción con una ontología, la búsqueda de información relacionada o similar dado un determinado concepto. En este caso concreto, nuestro sistema será capaz de obtener destinos equivalentes a Lanzarote, como puede ser cualquier otra isla del archipiélago canario.

- Para el caso del fragmento de la consulta "one week", el sistema es incapaz de extraer para el atributo "Duration" el valor de 7, ya que, como es evidente para nosotros, una semana equivale a siete días. Ésto es debido a que las reglas definidas sólo pueden reconocer patrones de texto en los que se encuentre la palabra clave "day", por lo que para el caso de "week" la información es ignorada, con la consecuente pérdida de precisión. Éste es otro de los problemas importantes a los que se enfrenta nuestro sistema.

Fase 2: Utilización de sinónimos

Si se observan detenidamente las limitaciones de la aproximación anterior, nos podemos dar cuenta de que si el desarrollador no añade al sistema sus propias reglas, incluyendo de esta forma conocimiento específico del dominio, éste no podrá realizar trabajo alguno durante las fases de análisis y extracción de información sobre las consultas textuales de los usuarios.

De esta forma, se puede ver que es importante que el sistema incorpore su propio conocimiento general e independiente de todo dominio para que pueda realizar trabajo aunque el usuario no proporcione sus propias reglas e informaciones.

Una forma de añadir este conocimiento general es utilizar **sinónimos**. Si el propio sistema busca los sinónimos de los valores contenidos en los atributos de cada caso de la base de casos y los compara con el contenido de las consultas textuales de los usuarios, podrá obtener información específica del dominio con el que se esté trabajando, pero de una forma general. De esta forma podemos salvar una de las mayores limitaciones encontradas en la primera aproximación de nuestro proyecto. Son, por tanto, los sinónimos la parte central y con más importancia de esta segunda aproximación.

La finalidad de la incorporación de la búsqueda de sinónimos a nuestro sistema es que sean utilizados de forma conjunta con las utilidades añadidas en la aproximación anterior, de forma que el análisis y posterior extracción de información de las consultas de texto se realiza de la manera más óptima.

La búsqueda de sinónimos en nuestro sistema se ha implementado mediante el uso de **WordNet**. WordNet es un diccionario o base de datos léxica en la que se agrupan las palabras en conjuntos de sinónimos llamados “synsets”. Además, proporciona definiciones cortas y generales, y especifica las relaciones semánticas existentes entre estos grupos de sinónimos. WordNet tiene como objetivo dos puntos: crear una estructura conjunta de diccionario y base de sinónimos con una utilización muy intuitiva, y permitir el análisis automático de textos y el desarrollo de aplicaciones en el campo de la Inteligencia Artificial.

Según se publica en la página de WordNet de la Universidad de Princeton, su desarrollo fue realizado por el Laboratorio de Ciencia del Conocimiento (en la propia Universidad de Princeton) bajo la dirección del profesor George A. Miller y la idea era buscar palabras conceptualmente, en lugar de alfabéticamente. La definición que se nos da en esa misma página acerca de esta aplicación es las siguiente “WordNet es un sistema de referencia léxica a través de la red, cuyo diseño está inspirado en las teorías psicolingüistas actuales acerca de la memoria léxica humana. Los nombres, verbos, adjetivos y adverbios están organizados en conjuntos de sinónimos y cada uno representa un concepto léxico subyacente. Diferentes relaciones enlazan los conjuntos de sinónimos”. Cabe destacar que el vocabulario de WordNet está originalmente en inglés, aunque se han hecho algunas traducciones a otros idiomas.

Si miramos otras definiciones sobre WordNet su sentido queda aún más claro. En la web de Elies (Estudios de Lingüística en Español) la definición que nos dan es la siguiente: “WordNet es una base de datos léxico-conceptual inglesa estructurada en forma de red semántica, es decir, compuesta de unidades léxicas y relaciones entre ellas, que pretende ser un modelo del conocimiento léxico-conceptual de los hablantes en inglés”. WordNet se basa en representar las que llaman categorías abiertas, es decir: nombres, adjetivos, verbos y adverbios. Las categorías cerradas (como pueden ser preposiciones, conjunciones, etc.) no son representadas en WordNet, ya que sus autores consideran que no son parte del conocimiento léxico-semántico, sino del conocimiento sintáctico.

Las bases de datos léxicas contienen una gran cantidad de información sobre elementos de tipo léxico: conceptos, significados, sinonimia, hiponimia, hiperonimia, meronimia... Como ya hemos dicho, la unidad básica de WordNet es el "sysnset" o conjunto de sinónimos (“conceptos asociados”), a través del cual se representan los conceptos. Las relaciones se establecen básicamente entre conceptos (y no entre palabras). Para la identificación de sinónimos se utiliza el contexto de la palabra.

La idea de utilizar bases de datos léxicas en sistemas de tratamiento de lenguaje natural no es nueva. En [02] se propone la utilización de WordNet para añadir más información a los sistemas de "Clasificación de Textos" (Text Categoristation), de manera que se mejore su rendimiento. En particular, en este texto se considera la opción de utilizar, precisamente, los "synsets" de WordNet para encontrar sinónimos de los nombres de las categorías y, como consecuencia, usarlos para predecir determinadas asignaciones. Además, propone utilizar las relaciones léxicas y conceptuales de WordNet para encontrar términos semánticamente relaciones con una cierta categoría. En una primera aproximación se utiliza la relación de sinonimia entre conceptos, aunque también se propone ampliar el sistema con otras relaciones como hiperonimia y meronimia.

La propuesta que hace esta referencia nos va a servir como ejemplo para el desarrollo del trabajo a realizar en esta aproximación de nuestro proyecto. Del mismo modo que se propone, se van a utilizar los "synsets" o conjuntos de sinónimos de WordNet para obtener información relativa que nos permita mejorar el rendimiento de nuestro sistema.

Para poder utilizar WordNet en nuestro proyecto Java sólo es necesario descargarse vía web una librería de libre distribución, llamada JWordNetLibrary (JWNL), en el que están contenidas todas las clases que se necesitan para poder utilizarlo. Después, hay que configurar este paquete para poder adaptarlo a nuestras necesidades. Esta configuración se realiza en un fichero XML, que también será leído por nuestro sistema para inicializar el uso del diccionario. Seguidos estos pasos, ya se pueden realizar todas las consultas que queramos.

El desarrollo de esta nueva utilidad en nuestro sistema nos otorga un gran avance a la hora de analizar las consultas textuales de los usuarios, pero plantea una nueva dificultad. Cuando los valores contenidos en los atributos de los casos del sistema son palabras reconocidas y existentes en WordNet todo funciona sin problemas, pero cuando estos valores son representaciones o expresiones específicas (por ejemplo, el valor "FiveStars" para el tipo enumerado "Accommodation") ésta funcionalidad ya no es útil, ya que este tipo de valores son "propios" de nuestro dominio específico.

Esta nueva dificultad plantea un nuevo y serio problema para nuestro sistema si no es tratado de forma adecuada. Para que esto no ocurra, en esta aproximación también hemos trabajado con **sinónimos definidos por el usuario**. De esta forma, los usuarios de la aplicación pueden definir sus propias palabras o expresiones mediante el uso de reglas en ficheros externos. Así, cada nueva característica definirá sus propios equivalentes o sinónimos, tal y como si se tratara de un diccionario personal. Además, y para añadir una mayor efectividad en este trabajo, en cada una de este nuevo tipo de reglas se puede especificar el atributo de la estructura de casos en el que tienen vigencia estas similitudes, es decir, su ámbito.

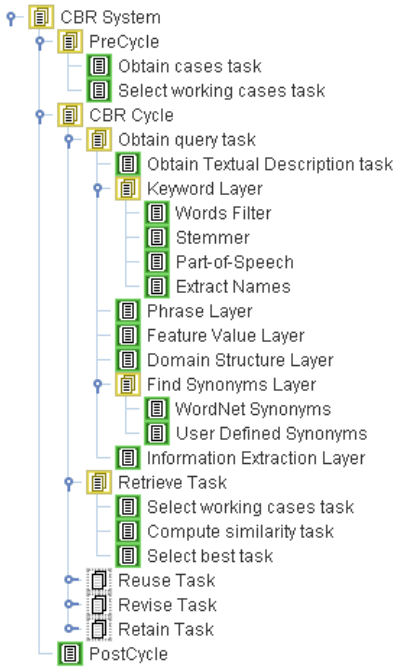
Para poder llevar a cabo la utilidad necesario ha sido necesario crear un nuevo tipo de reglas, que serán explicadas con más detalle en los próximos apartados de esta sección.

Con estas dos nuevas utilidades añadidas a nuestro sistema, hemos conseguido limitar varias de las limitaciones de la aproximación anterior. Así, aunque el usuario no añada conocimiento específico del dominio (en forma de reglas) el sistema puede obtenerlo de los valores contenidos en cada uno de los casos de la base de casos. También de esta forma se está dotando al sistema de una mayor flexibilidad a la hora de reconocer palabras y expresiones en las consultas textuales de los usuarios, ya que el uso de los sinónimos, tanto los encontrados en WordNet como los definidos por el usuario, nos lo permite.

Al igual que en la sección anterior, en el siguiente apartado se explicarán en detalle las nuevas utilidades añadidas a nuestro sistema CBR a través de las etapas de éste. Como la mayoría de las capas de esta nueva aproximación son iguales que en la anterior, su explicación se obviará.

Etapas del sistema

En la figura que se muestra a continuación se puede ver la estructura en capas del sistema para esta segunda fase de desarrollo. La única diferencia entre la estructura actual con la de la fase anterior es la inclusión de la nueva capa "Synonyms Layer", subdividida en dos etapas: una para los sinónimos obtenidos de WordNet y otra para los sinónimos definidos por el usuario. Además, ha sido necesario incluir en la etapa de Pre-ciclo una nueva tarea de selección de casos, para que éstos puedan ser accedidos y utilizados por la nueva capa de sinónimos.



A continuación se introducen brevemente las acciones que se realizan en cada una de las nuevas etapas añadidas a nuestro sistema. En los siguientes apartados se explica más en detalle cada una de estas capas.

- **Pre-ciclo** (PreCycle). En esta etapa se ha añadido una nueva sub-tarea que se encarga de seleccionar los casos de trabajo de la base de conocimiento. Esta tarea ya se realizaba en la fase anterior del sistema, pero ahora es necesario realizarla antes de la nueva capa de sinónimos.
    - Tarea de obtención de casos (Obtain cases task).
    - **Tarea de selección de casos activos** (Select working cases task). El sistema selecciona cuáles serán los casos que van a intervenir en la tarea de recuperación. En nuestro caso, y generalmente siempre, se seleccionan todos los que forman parte de la base de conocimiento.
- **Ciclo CBR** (CBR Cycle).
    - **Tarea de obtención de la query** (Obtain Query Task). En esta etapa ha sido incluida una nueva sub-tarea con la utilidad desarrollada en esta fase del proyecto.
        - Tarea de obtención de la consulta textual (Obtain Textual Description task).
        - Capa de palabras clave (Keyword Layer).

        - Filtro de palabras (Words Filter).
        - Extracción de raíces (Stemmer).
        - Categorías léxicas (Part-of-Speech).
        - Extracción de nombres (Extract Names).

    - Capa de expresiones propias (Phrase Layer).
    - Capa de características (Feature Value Layer).
    - Capa de estructura de dominio (Domain Structure Layer).

    - **Capa de sinónimos** (Synonyms Layer). Esta capa divide su trabajo en dos sub-tareas que realizan búsquedas de sinónimos en la consulta textual del usuario, con el objetivo de encontrar una mayor cantidad de datos relevantes.

        - **Sinónimos WordNet** (WordNet Synonyms). Esta etapa del sistema busca sinónimos de las palabras que forman parte de las consultas de los usuarios y verifica si éstos coinciden con valores contenidos en la base de conocimiento del CBR.

        - **Sinónimos definidos por el usuario** (User Defined Synonyms). Esta capa encuentra en la consulta del usuario sinónimos definidos por el propio desarrollador, mediante la utilización de una serie de reglas incluidas en un fichero de configuración

    - Capa de extracción de información (Information Extraction Layer).

- Tarea Recuperar (Retrieve Task).

    - Tarea de selección de casos activos (Select working cases task).
    - Tarea de cómputo de similitud (Compute similarity task).
    - Tarea de selección del mejor (Select best task).

- Tarea Reutilizar (Reuse Task). Esta tarea no se utiliza.
- Tarea Revisar (Revise Task). Esta tarea no se utiliza.
- Tarea Recordar (Retain Task). Esta tarea no se utiliza.

- Post-ciclo (PostCycle). A esta etapa no se han añadido nuevas tareas.

Pre-ciclo (PreCycle)

Localización: PreCycle (etapa principal).
Clase Java que implementa la tarea: jcolibri.extensions.user.method.ObtainAndSelectWorkingCases.
Parámetros: ninguno.
Tipo de tarea: descomposición.

La etapa de Pre-ciclo de nuestro sistema ha tenido que ser cambiada para añadirle la tarea de selección de casos "Select working cases task". Esta capa, como ya explicamos en la sección anterior, se encarga de añadir al contexto del sistema los casos que van a ser utilizados por éste. Recordamos que para el dominio específico de los viajes se toman todos los casos existentes.
Esta acción es necesario realizarla en la primera etapa del sistema CBR porque la nueva capa de sinónimos necesita poder acceder a los casos y trabajar con ellos y con los valores que éstos contienen. Con esta nueva etapa en el Pre-ciclo del sistema logramos este comportamiento.

Capa de sinónimos (Synonyms Layer)

Localización: CBR Cycle → Synonyms Layer.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.FindSynonymsMethod.
Parámetros: ninguno.
Tipo de tarea: descomposición.

Esta nueva tarea de descomposición contiene las dos etapas de búsqueda de sinónimos del sistema. Su único objetivo es el de contener a estas dos nuevas capas.

Sinónimos WordNet (WordNet Synonyms)

Localización: CBR Cycle → Synonyms Layer → WordNet Synonyms.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.WordNetSynonymsMethod.
Parámetros: ninguno.
Tipo de tarea: resolución.

Esta etapa se encarga de realizar una búsqueda de sinónimos entre las palabras que componen la consulta textual del usuario con la finalidad de encontrar coincidencias entre éstos y los valores que componen la base de casos.

La primera operación que realiza este método es la de extraer todos los atributos de tipo cadena de texto que encuentre en toda la base de casos del sistema. Para ello, toma de uno en uno cada caso y mira en todos sus atributos en busca de valores no nulos. Este trabajo es realizado por una pequeña función que hemos diseñado con tal finalidad, y cuyo resultado es una lista de objetos del tipo "SimpleAttribute" (definido por nosotros), cada uno de los cuales almacena la siguiente información:

- **Descripción.** Nombre del atributo.
- **Valor.** Valor que contiene el atributo. Puede ser nulo.

Una vez extraídos todos los valores no nulos encontrados en la base de casos del sistema CBR, es necesario procesar el texto correspondiente a la consulta realizada por el usuario, y que está disponible en el contexto del sistema. Lo primera acción a realizar con esta estructura es la extracción de los textos que la componen, que, según ya se ha explicado, sólo está formado por uno (consulta del usuario), aunque su implementación se ha realizado de forma dinámica por si en un futuro próximo se amplia esta funcionalidad a un mayor número de textos.

A partir del texto obtenido, es necesario extraer cada una de las palabras que lo conformar, siempre que no se trate de "palabras de parada". Para conseguir este comportamiento es necesario definir un filtro y utilizarlo cada vez que deseemos extraer información de cada uno de los párrafos que componen el texto. Para definir un filtro sólo es necesario crear un objeto de tipo "Hashtable" con las propiedades que se requieren ser cumplidas, en este caso, no ser palabra de parada, por lo que la única propiedad existente será la definida para el atributo "Token.ISNOT_STOPWORD" con un valor de "true", filtro que sólo pasarán las palabras que cumplan dicha propiedad (no ser palabra de parada).

Para poder extraer las palabras relevantes que forman parte del texto de consulta es necesario ir descomponiéndolo en sub-partes: primero en párrafos ("Paragraph"), después en oraciones ("Sentence") y, por último, en tokens ("Token").

Tras descomponer todo el texto de consulta en tokens (párrafo a párrafo), es cuando entran en juego los sinónimos de WordNet. Para realizar una búsqueda de sinónimos en este diccionario hemos diseñado una función que recibe los siguientes parámetros:

- **Palabra a buscar (token).** Se corresponde con la palabra sobre la que se desean buscar los sinónimos, y que se almacena en el atributo COMPLETEWORD del token correspondiente.

- **Categoría léxica de la palabra (POS).** Como ya se explicó en el apartado correspondiente a la primera aproximación, la etapa "Part-of-Speech" es la encargada de asignar a cada palabra (que no sea de parada) la categoría léxica a la que corresponde. Este valor se encuentra disponible en el atributo POSTAG del token correspondiente. Las categorías con las que se trabaja el diccionario WordNet son las siguientes:

  - Nombre.
  - Verbo.
  - Adejetivo
  - Adverbio.

  por lo que es necesario realizar una sencilla transformación sobre las categorías asignadas por la etapa "Part-of-Speech", generalizando éstas. Por ejemplo, si la categoría asignada en la etapa POS es "Nombre Común", la categoría final para WordNet será simplemente "Nombre".

- **Lista para almacenar los valores de retorno.** En esta lista se almacenarán todos los sinónimos para la palabra sobre la que se están buscando.

- **Tabla Hash para evitar palabras repetidas.** Esta tabla se encarga de asegurar que no se almacenan valores repetidos en la lista anterior.

Esta función se encarga de realizar peticiones al diccionario de WordNet, tras haberlo inicializado con un fichero XML con unos parámetros específicos de configuración. Estas peticiones se realizan preguntando por los "Synsets" de la palabra que se está pasando como parámetro. Estos "Synsets" son las definiciones que cualquier diccionario nos da de una cierta palabra. Pero lo que nosotros buscamos en este caso son sólo palabras, por lo que los valores devueltos serán las palabras representantes de estas definiciones y que WordNet nos permite acceder a ellas mediante el acceso a sus atributos "Lemma".

Tras tener todos los sinónimos de los tokens del párrafo actual es necesario que el sistema busque coincidencias con los valores de los casos extraídos al comienzo del método. Para realizar esta búsqueda también hemos diseñado una pequeña función que requiere los siguientes parámetros:

- Lista de sinónimos.
- Lista con todos los valores no nulos de la base de casos.
- Lista para almacenar las coincidencias encontradas.

Esta función recorre ambas listas (la de sinónimos y la de valores de los casos) y va comparando uno a uno si son iguales, sin tener en cuenta las mayúsculas y minúsculas. En caso de que sean iguales, se añadirá el valor del caso a la última lista, ya que es éste el representante en el sistema y el que, en caso de tratarse de un valor de un tipo enumerado, el que se corresponda con él. Es decir, el valor del caso representa un cierto valor dentro del sistema que agrupa a un grupo de palabras que son sinónimos.

En este caso, también se ha considerado la posibilidad de que el sinónimo y el valor del caso no coincidan exactamente, sino que el primero esté contenido en el segundo, caso que se da al tratarse de diferentes conjugaciones de un mismo verbo. En este caso, se procederá al contrario, obteniendo los sinónimos del valor del caso y comparando su igualdad con el sinónimo de la query original. De esta forma, se proporciona una sencilla forma de averiguar si dos verbos son en realidad el mismo pero con diferentes conjugaciones, ya que al calcular los sinónimos de una forma conjugada, uno de los resultados devueltos es el infinitivo del verbo. Si realizamos esta operación con ambos valores (sinónimo y valor del caso) obtendremos valores idénticos.

Por ejemplo, en el tipo enumerado "HolidayType" existe el valor "Skiing". Si el usuario introduce en la query un fragmento con el verbo "ski", por ejemplo "I would like to ski in the mountains", al llegar al token "ski" el sistema calculará sus sinónimos, que al tratarse de un verbo devolverá como uno de sus valores el propio infinitivo, que en este caso coincide. Al comparar con el valor "Skiing" el sistema detectará que el sinónimo está contenido en él, pero no son iguales, por lo que calculará los sinónimos de "Skiing", obteniendo como primer valor el infinitivo del verbo, "Ski", que posteriormente comparará con el primer sinónimo y determinará que ambos son iguales. Por tanto, el valor "Skiing" será almacenado como valor del atributo "HolidayType" para la query del usuario.

Tras realizar una búsqueda de coincidencias entre los sinónimos de las palabras del párrafo en proceso y todos los valores extraídos de la base de casos, es necesario almacenar la información encontrada. Como la función anterior nos devuelve una lista con todas las coincidencias, el único trabajo necesario será recorrerla e ir asociando está información al párrafo mediante el uso de objetos "FeatureInfo", de igual modo y como ya se vio en la capa "Feature Value Layer". Recordemos que esta información almacenada en los párrafos de forma temporal será extraída próximamente por la capa de extracción de información y almacenada en los atributos de la nueva query estructurada que se cree. En este paso también se realiza una comprobación, con la ayuda de una tabla Hash, para evitar el almacenamiento de valores repetidos.

Sinónimos definidos por el usuario (User Defined Synonyms)

Localización: CBR Cycle → Synonyms Layer → User Defined Synonyms.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.UserDefinedSynonymsMethod.
Parámetros: procesar query (de tipo booleano), procesar casos (de tipo booleano) y fichero donde el usuario ha definido, en forma de reglas, sus propios sinónimos (de tipo File).
Tipo de tarea: resolución.

El comportamiento de este etapa es muy parecido al ya descrito en la capa "Feature Value Layer", ya que en esta también se utiliza un fichero externo en el que el usuario ha definido sus propios sinónimos. La finalidad de esta tarea es la búsqueda de estos sinónimos en los textos de consultas de los distintos usuarios, de forma que el sistema almacene esta información en la query estructurada.

Este método se ha diseñado para que pueda ser aplicado tanto a la query textual del sistema como a todos los casos que forman parte de la base de casos, por eso son necesarios dos atributos booleanos que indican que estructuras han de ser incluidas en el análisis, pudiendo ser ambas (query y casos).

La primera acción que ha de realizar este método es la lectura del fichero donde el usuario ha definido sus propios sinónimos mediante el uso de un nuevo tipo de reglas diseñado por nosotros y que a continuación se describen:

[FeatureName]{FeatureValue}FeatureRegularExpresion

- FeatureName: nombre del atributo de la estructura del caso dónde tienen vigencia los sinónimos definidos en la expresión, es decir, su "ámbito".
- FeatureValue: concepto o valor representante de todos los atributos. Suele ser un valor contenido en un tipo enumerado.
- FeatureRegularExpression: expresión definida siguiendo la sintaxis que se especifica en la clase Java java.util.regex.Pattern.

Ejemplo: [Accomodation]{OneStar}(one stars?)|(1 stars?)|(one-star)|(1-star)|([Hh]ostel)
|([Bb]udget [Hh]otels?)

La interpretación de este tipo de reglas es el siguiente. Si algún fragmento el texto de consulta del usuario cumple la expresión definida en la regla, se almacenará el valor "FeatureValue" en el atributo de la query con el nombre "FeatureName".

Por ejemplo, en el texto “I want a hotel with just one star” el sistema debe reconocer el fragmento “one star”, ya que este está definido dentro de la expresión de la regla de ejemplo anterior, por lo que, como siempre, almacenará temporalmente la información relativa en el párrafo (con un objeto FeatureInfo), con un valor “OneStar” para el atributo “Accommodation”.

Como parte del trabajo realizado en esta aproximación, se han definido una serie de reglas para probar el buen funcionamiento de esta nueva etapa del sistema. Estas reglas son las siguientes:

```
[Accommodation]{FiveStars}([Ff]ive [Ss]tars?)|(5 [Ss]tars?)|([Ff]ive-[Ss]tar)
|(5-[Ss]tar)|([Ll]uxury)|([Ll]uxurious)|([Ss]uite)|([Rr]esort)|([Dd]eluxe)

[Accommodation]{FourStars}([Ff]our [Ss]tars?)|(4 [Ss]tars?)|([Ff]our-[Ss]tar)
|(4-[Ss]tar)|([Ss]emi[Ll]uxury)|([Ss]emi-[Ll]uxury)|([Ss]uite)|([Rr]esort)
|([Dd]eluxe)

[Accommodation]{HolidayFlat}([Hh]oliday[Fl]lats?)([Hh]oliday [Aa]partments?)
|([Ii]nn)

[Accommodation]{ThreeStars}(three stars?)|(3 stars?)|(three-star)|(5-star)
|([Mm]id [Pp]riced [Hh]otels?)|([Mm]id [Rr]ange)|([Mm]id-[Rr]ange)

[Accommodation]{TwoStars}(two stars?)|(2 stars?)|(two-star)|(2-star)
|([Bb]udget [Hh]otels?)

[Accommodation]{OneStar}(one stars?)|(1 stars?)|(one-star)|(1-star)|([Hh]ostel)
|([Bb]udget [Hh]otels?)

[Season]{January}([Ww]inter)
[Season]{February}([Ww]inter)
[Season]{March}([Ss]pring)
[Season]{April}([Ss]pring)
[Season]{May}([Ss]pring)
[Season]{June}([Ss]ummer)
[Season]{July}([Ss]ummer)
[Season]{August}([Ss]ummer)
[Season]{September}([Aa]utumn)|([Ff]all)
[Season]{October}([Aa]utumn)|([Ff]all)
[Season]{November}([Aa]utumn)|([Ff]all)
[Season]{December}([Ww]inter)|([Cc]hristmas)|([Hh]olidays)

[HolidayType]{Bathing}([Bb]each)|([Ss]ea)|([Ss]wim)|([Ss]wimming)
|([Ss]wimming-pool)|([Ss]wimming pool)

[HolidayType]{Active}([Aa]ction)|([Aa]dventure)|([Ee]xcursion)
[HolidayType]{City}([Mm]useums?)|([Mm]onuments)|([Gg]uided [Tt]ours?)
|([Rr]estaurants?)|([Ss]ightseeing)|([Tt]heatres?)

[HolidayType]{Recretation}([Ll]eisure)|([Cc]inemas?)|([Mm]ovies?)|([Tt]heatres?)
[HolidayType]{Wandering}([Ww]alk)|([Ww]alking)
[HolidayType]{Language}([Ss]peak)|([Ss]peaking)
[HolidayType]{Education}([Ss]chool)|([Uu]niversity)|([Ll]essons)|([Cc]lasses)
|([Mm]useums?)

[HolidayType]{Skiing}([Ss]now)|([Ss]nowboard)|([Ss]nowboarding)|(Slalom)

[Region]{Cairo}([Ee]l [Cc]airo)
[Region]{ErzGebirge}([Ee]rz [Gg]ebirge)
[Region]{NorthSea}([Nn]orth [Ss]ea)
[Region]{BalticSea}([Bb]altic [Ss]ea)
[Region]{BlackForest}([Bb]lack [Ff]orest)
[Region]{CotedAzur}([Cc]oted [Aa]zur)
[Region]{AdriaticSea}([Aa]driatic [Ss]ea)
[Region]{LakeGarda}([Ll]ake [Gg]arda)
[Region]{SalzbergerLand}([Ss]alzberger [Ll]and)
```

```
[Region]{CostaBlanca}([Cc]osta [Bb]lanca)
[Region]{CostaBrava}([Cc]osta [Bb]rava)
[Region]{GranCanaria}([Gg]ran [Cc]anaria)
[Region]{Teneriffe}([Tt]enerife)
[Region]{GiantMountains}([Gg]iant [Mm]ountains)
[Region]{TurkishAegeanSea}([Tt]urkish [Aa]egean [Ss]ea)
[Region]{TurkishRiviera}([Tt]urkish [Rr]iviera)
[Region]{Czechia}([Cc]zech)
```

Según ya se ha explicado, el primer paso necesario es procesar el fichero donde se encuentran definidas todas estas reglas, y cuya ruta se pasa como parámetro a este método. Hemos diseñado una función que procese este fichero, con una estructura muy similar a la ya proporcionada en las clases que también trabajan con este tipo de ficheros.

Una vez determinadas las estructuras a analizar (query, casos o ambos), es necesario ir procesándolas una a una, extrayendo para ello los textos que contienen. La idea inicial de este método es analizar sólo la query, que como sabemos sólo contiene un texto con la consulta del usuario, aunque la implementación se ha realizado de forma genérica para aceptar todo tipo de posibilidades. En resumen, la búsqueda de sinónimos se realizará en los textos contenidos en las estructuras que forman parte del proceso.

Para realizar el análisis pertinente en búsqueda de sinónimos "propios" es necesario aplicar cada una de las reglas definidas en el fichero externo, y que, después de ser extraídas, ya se encuentran disponibles para poder ser usadas. Por tanto, para cada texto se aplicarán todas las reglas definidas por el usuario.

La información contenida dentro de la estructura de cada una de estas reglas ("FeatureName", "FeatureValue" y "FeatureRegularExpression") es almacenada en objetos que nosotros hemos definido, de nombre "Triple" y cuyo contenido es el siguiente:

- **Feature.** Nombre del atributo. Correspondiente al "FeatureName" de la regla.
- **Value.** Valor a almacenar en el atributo. Correspondiente al "FeatureValue" de la regla.
- **Pattern.** Expresión de reconocimiento. Correspondiente a la "FeatureRegularExpression".

Esta distribución facilitará la búsqueda de sinónimos propios y dotará al sistema de una mayor eficacia y simplicidad.

Para poder aplicar cada una de estas reglas es necesario descomponer cada texto en párrafos, y obtener de ellos su contenido, que se encuentra disponible en el atributo RAW_DATA de su estructura correspondiente (Paragraph). Una vez hecho ésto, se aplicará a cada fragmento de texto el "pattern" definido en la regla, mediante el uso de una estructura "Matcher" proporcionada por Java, que busca a ver si la expresión se cumple y, en ese caso, qué fragmento del texto es el que hace que se cumpla. Esta estructura también proporciona una sencilla forma de ver si ha habido alguna coincidencia. En el caso de que las haya habido es necesario almacenar la información relativa de forma asociada al párrafo en proceso, mediante el uso de un objeto "FeatureInfo", como ya se ha explicado otras veces.

El valor que se almacenará no será el encontrado en el texto de consulta del usuario, sino el indica en el campo "FeatureValue" de la regla. También será necesario indicar el nombre del atributo en el que se almacenará este valor, definido en el campo "FeatureName" de la regla. En la capa de extracción de información será donde se extraigan estos datos asociados a los textos y se almacenen de forma permanente en la nueva query estructurada que se construya.

Como última apreciación de este apartado, se puede decir que en la estructura de las reglas se podría haber obviado el campo "FeatureName", de forma que cada vez que se encontrara en el texto de consulta un fragmento que cumpliera la expresión, se buscara entre todos los valores de la base de casos

del sistema el valor definido en la parte "FeatureValue", para hallar así el nombre del atributo donde se habría de almacenar este valor (mediante coincidencia de valores entre query y caso).

Al integrar este campo como parte de la estructura de la regla ahorramos una gran cantidad de trabajo al sistema, ya que la extracción de información se realiza de forma inmediata, definiendo en la regla el nombre del atributo donde se ha de almacenar el valor correspondiente. Así, la eficiencia de todo el sistema CBR aumenta, mientras que la complejidad de las reglas y de esta etapa en concreto no se ven casi afectadas.

Nuevas utilidades añadidas

A lo largo de esta segunda aproximación de nuestro proyecto hemos añadido nuevas funcionalidades a nuestro sistema, que mejoran sustancialmente el comportamiento de la anterior, además de proporcionar soluciones a las limitaciones más serias de éstas.

Las principales utilidades añadidas en esta etapa son las siguientes:

- Adición de nuevo conocimiento al sistema mediante el uso de sinónimos de WordNet.
- Posibilidad de definición de sinónimos propios.

Adición de nuevo conocimiento al sistema mediante el uso de sinónimos de WordNet

En la primera aproximación del proyecto destacábamos que la principal limitación del sistema era la incorporación de conocimiento específico del dominio por parte del usuario. Si éste no proveía las reglas necesarias, el sistema no realizaría trabajo alguno y no podría dar servicio al usuario. Este problema desaparece con el trabajo realizado en esta segunda aproximación, gracias al uso de los sinónimos de diccionario WordNet.

Con la utilización de este recurso se otorga al sistema CBR de una independencia sobre el conocimiento específico del dominio que ha de ser proporcionado por el usuario de la herramienta, ya que la nueva etapa añadida se encargará de buscar información, mediante el uso de sinónimos, por todos valores contenidos en la base de casos.

Así, aunque el usuario no incorpore sus propias reglas, el sistema podrá realizar trabajo para intentar inferir que información es relevante dentro de una consulta de texto.

Posibilidad de definición de sinónimos propios

Una de las limitaciones de la primera aproximación de nuestro proyecto era la utilización de valores concretos o "conceptos" para representar ciertas palabras o expresiones dentro de un dominio. Recordemos por ejemplo el caso del tipo enumerado "AccommodatiosEnum", con valores "OneStar, "TwoStars", etc. Estos valores podían ser reconocidos por el sistema pero lo más probable es que el usuario los desconociera o que no utilizara exactamente las mismas expresiones para referirse al mismo concepto.

Es por esta razón por la que en esta segunda etapa se ha provisto al sistema de un mecanismo que permite la definición de sinónimos propios, es decir, conjunto palabras o expresiones que representan un mismo concepto. De esta forma, se permite una mayor flexibilidad a la hora de analizar y extraer información de las consultas textuales de los usuarios del sistema.

Limitaciones del sistema

A pesar de que con el trabajo realizado en esta segunda aproximación se han resuelto algunas de las limitaciones de la primera etapa del proyecto, el uso de la tecnología de sinónimos también tiene sus limitaciones y presenta algunas dificultades con las que es necesario convivir. A continuación se enumeran algunas de esas limitaciones:

- Posibilidad de existencia de palabras y expresiones ambiguas.
- Mayor cantidad de información procesada introduce un mayor retardo en el sistema.

Algunas de las limitaciones que aún permanecen en el sistema son:

- El sistema no permite la representación de negaciones, conjunciones y disyunciones.
- El sistema no permite representar atributos con valores múltiples.
- El usuario no puede confirmar sus datos.
- El sistema no es capaz de obtener información relativa a un determinado concepto.

Posibilidad de existencia de palabras y expresiones ambiguas

La ambigüedad semántica entre palabras y expresiones siempre ha sido un problema dentro del análisis de textos, por lo que en esta ocasión no podía ser diferente. Por ejemplo, la palabra "banco" puede referirse al lugar dónde las personas se sientan, a las entidades relacionadas con el dinero o un conjunto de peces. Como ya hemos visto antes, el uso de los sinónimos de WordNet otorga al sistema una mayor flexibilidad, pero puede introducir ciertos problemas si el sistema encuentra ambigüedad.

En la etapa de búsqueda de sinónimos el sistema busca coincidencias entre las palabras de la consulta textual del usuario y los valores contenidos en cada uno de los casos de la base de casos. Si estas palabras tienen varios significados (polisemia) el sistema puede interpretar un significado incorrecto y almacenar la información relativa, de forma que los resultados obtenidos no sean los esperados.

Aunque se trabaja con un dominio muy reducido, siempre existe la posibilidad de encontrar palabras y expresiones ambiguas, que alteren el funcionamiento correcto del sistema.

Mayor cantidad de información procesada introduce un mayor retardo en el sistema

En la etapa de búsqueda de sinónimos de WordNet es necesario obtener todos los valores de cada uno de los casos de la base de casos del sistema, de forma que éstos puedan ser comparados con los sinónimos de las palabras contenidas en la consulta textual del usuario. De esta forma, a mayor cantidad de información a procesar, mayor será el retardo del sistema.

En el caso concreto del dominio de los viajes, la base de casos está compuesta por 1024 casos, cada uno de los cuales está formado por 9 atributos, que han de ser procesados todos para determinar si la información que contienen es o no útil. Además, también hay que obtener para todas las palabras de la consulta, que no sean de "parada", sus sinónimos. Por último, es necesario ir comparando todos estos datos con los valores de la base de casos, uno a uno, para determinar si existen coincidencias. Todo este procesamiento de información incluye en el sistema un retardo significante.

Problemas encontrados

Al igual que en la anterior aproximación de nuestro proyecto, en esta etapa han surgido una serie de problemas que hemos tenido que ir resolviendo poco a poco. Éstos son algunos de esos problemas:

- Aprendizaje de JWordNet.
- Duda sobre qué valores se han de buscar los sinónimos.
- Almacenamiento de la información sobre sinónimos extraída.
- Palabras y expresiones ambiguas.
- Selección de los casos que formarán parte de la base de casos antes de su utilización.

Aprendizaje de JWordNet

Como es natural, para poder a comenzar a utilizar las librerías de JWordNet fue necesario un estudio previo de su funcionamiento, por lo que ésto supuso una primera dificultad a la hora de comenzar con nuestro trabajo en esta etapa del proyecto.

Para obtener información acerca de su funcionamiento consultamos ayudas y tutoriales de varias webs, así como la página oficial donde es posible descargar el paquete con el código necesario para poder utilizar el diccionario de WordNet. En dicha página nos fueron muy útiles los foros, en los que muchos usuarios también realizan peticiones de ayuda en busca de ejemplo de iniciación.

Duda sobre qué valores se han de buscar los sinónimos

Cuando ya supimos cómo utilizar el diccionario de WordNet, nuestra siguiente dificultad se planteó a la hora de definir sobre qué valores tendríamos que buscar los sinónimos, con dos posibles opciones:

- Sobre las palabras que componen la consulta textual del usuario.
- Sobre los valores de los casos del sistema.

Como ya se ha explicado en este mismo apartado, la opción elegida fue la primera. De esta forma, del texto de consulta del usuario se extraen las palabras que no sean de "parada" y se buscan sus sinónimos, comparando éstos después con todos los valores que forman parte de la base de casos.

En el caso de que hubiéramos elegido la segunda opción, la cantidad de procesamiento del sistema sería mucho mayor, ya que habría que buscar un mayor número de sinónimos (por cada valor encontrado en la base de casos), aunque está opción también habría podido ser completamente viable.

Otro problema surgió con los verbos y sus conjugaciones sobre los que se obtienen sus sinónimos. Como hemos visto en anteriores apartados, es necesario realizar un trabajo adicional para determinar que una cierta forma verbal procede de un determinado verbo.

Almacenamiento de la información sobre sinónimos extraída

Otra duda surgió a la hora de almacenar la información determinada como relevante, es decir, cuando encontramos una coincidencia entre un sinónimo y un valor de la base de casos. Estos datos han de ser almacenados de forma temporal como parte de la consulta textual, utilizando para ello objetos de un cierto tipo.

Para aprovechar al máximo la infraestructura que nos otorga el entorno jCOLIBRI, se han utilizado objetos del tipo "FeatureInfo", al igual que en la capa "Feature Value Layer". Otra opción posible habría sido utilizar un nuevo tipo de objetos definido específicamente para este tipo de características, de forma que almacenara el nombre del atributo, su valor y el fragmento del texto equivalente, y que hubiera servido tanto para la etapa de búsqueda de sinónimos en WordNet como en la correspondiente a los sinónimos definidos por el usuario.

Palabras y expresiones ambiguas

Como ya hemos explicado, la ambigüedad entre palabras y expresiones constituye una serie limitación a la hora de realizar un análisis y posterior extracción de información relevante sobre un texto. En esta etapa del proyecto nos encontramos con este problema, que en ciertos momentos resulta desconcertante.

En ocasiones nuestro sistema ha errado debido a la ambigüedad de palabras o expresiones, como por ejemplo, en el dominio específico de los viajes, determinar una palabra como un cierto verbo cuando en realidad forma parte del nombre de un hotel.

Para evitar algunas de estas situaciones ha hecho realizar algunos cambios al trabajo inicial de esta etapa, de forma que, por ejemplo, sólo se busquen datos para valores simples, y no para expresiones completas o cadenas de texto (como nombres de hoteles).

Selección de los casos que formarán parte de la base de casos antes de su utilización

Un pequeño problema que tuvimos en esta etapa del proyecto fue el desconocimiento de que, para poder utilizar y trabajar con los casos que forman parte de la base de casos del sistema es necesario seleccionarlos e incluirlos explícitamente en éstos, ya que no basta con que el conector que opera en el Pre-ciclo los extraiga de la base de datos.

El problema que encontrábamos era que, al acceder a la base de casos y extraer todos los casos, se obtenía un error en el que se indicaba que no existía ninguno, ya que estos no se habían añadido explícitamente.

Ésta es la razón por la cual se ha modificado esta primera etapa de nuestro sistema CBR, de forma que se extraigan los datos del recurso de persistencia, se conviertan en casos y se añadan a la base de casos para que puedan ser utilizados por las posteriores tareas.

Pruebas realizadas

En este apartado de la segunda fase de desarrollo de nuestro sistema se detallan algunas de las pruebas que se han llevado a cabo contra el sistema para comprobar su funcionamiento. En este caso se va a mostrar principalmente la nueva funcionalidad añadida de los sinónimos, mediante su utilización en varios ejemplos representativos. Como ya hemos explicado a lo largo de este apartado, la nueva etapa creada busca en las consultas de los usuarios sinónimos, tanto en WordNet como los definidos por el propio desarrollador, para mejorar el proceso de extracción de información.

En las siguientes consultas de ejemplo se subrayan los conceptos relevantes que deberían ser extraídos por nuestro sistema.

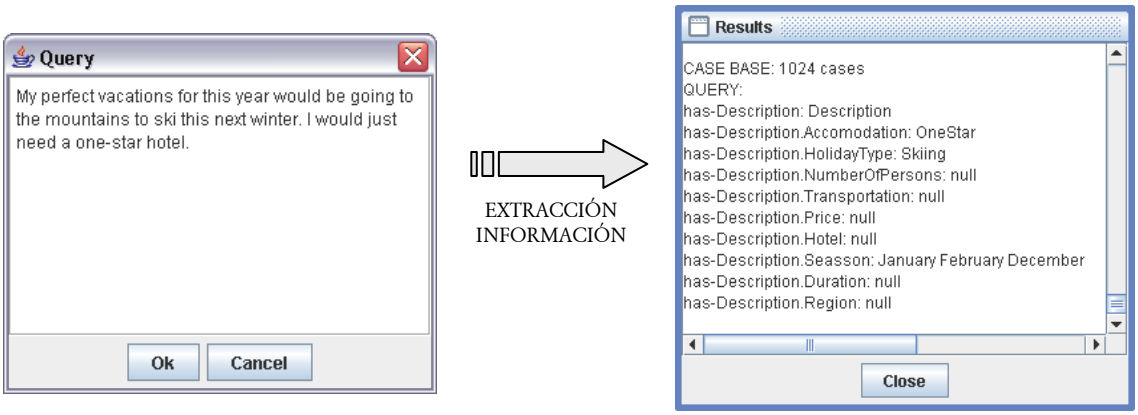
Prueba 1

Consulta

- My perfect vacations for this year would be going to the mountains to ski this next winter. I would just need a one-star hotel.

Información extraída

- Accommodation = OneStar
- HolidayType = Skiing
- Season = January February December



En esta primera consulta de prueba se extraen tres conceptos de la consulta textual en la nueva etapa de sinónimos incluida en esta fase. En primer lugar, el sistema se encarga de obtener los sinónimos con WordNet de las palabras relevantes de la consulta (que no sean palabras de parada...), entre las que se incluye "ski". Para ella, el sistema busca sus sinónimos y comprara con los datos de la base de casos para comprobar si existen coincidencias. Como se trata de un verbo, la capa de WordNet devuelve todos sus sinónimos más el infinitivo del mismo, que en este caso coincide. Uno de los valores de la enumeración del atributo "HolidayType" es "Skiing", que contiene la propia palabra "ski". Para asegurarse de que ambas cadenas se refieren a lo mismo (acción de esquiar), la capa de WordNet obtiene los sinónimos de "Skiing", entre los que se encuentra "ski". Por tanto, "ski" es considerado sinónimo de "Skiing" y, por tanto, esta información es almacenada como dato relevante del texto de consulta.

Una vez realizadas las acciones anteriores en la capa de WordNet, el sistema comienza con la etapa de sinónimos definidos por el usuario, en la que se utiliza un fichero de configuración externo en el que el

desarrollador del sistema define una serie de reglas para la identificación de conceptos mediante palabras o expresiones propias del lenguaje o del dominio. En nuestro caso específico, hemos incluido las siguientes reglas, que son utilizadas en este ejemplo:

```
[Accomodation]{OneStar}(one stars?)|(1 stars?)|(one-star)|(1-star)
|([Hh]ostel)|([Bb]udget [Hh]otels?)
```

```
#[Seasson]{January}([Ww]inter)
#[Seasson]{February}([Ww]inter)
#[Seasson]{December}([Ww]inter)|([Cc]hristmas)|([Hh]olidays)
```

La primera de ellas identifica el concepto "OneStar" con múltiples expresiones que permiten expresar lo mismo: hoteles de una estrella. Al aplicar este patrón de regla sobre el texto de consulta, es posible identificar el fragmento "one-star" como expresión propia para el valor "OneStar", que, por tanto, será almacenado como dato para el atributo "Accommodation". En el caso de las últimas reglas, se definen sinónimos para cada uno de los meses del año, valores que forman parte de la enumeración del atributo "Season". Concretamente, el objetivo es identificar las estaciones del año con los meses en los que tienen lugar. De esta forma, con esta regla es posible identificar en el texto de consulta el fragmento "winter" y considerar como sinónimos los valores "January", "February" y "December", meses en los que transcurre el invierno, y almacenarlos en el atributo "Season" como información relevante extraída.

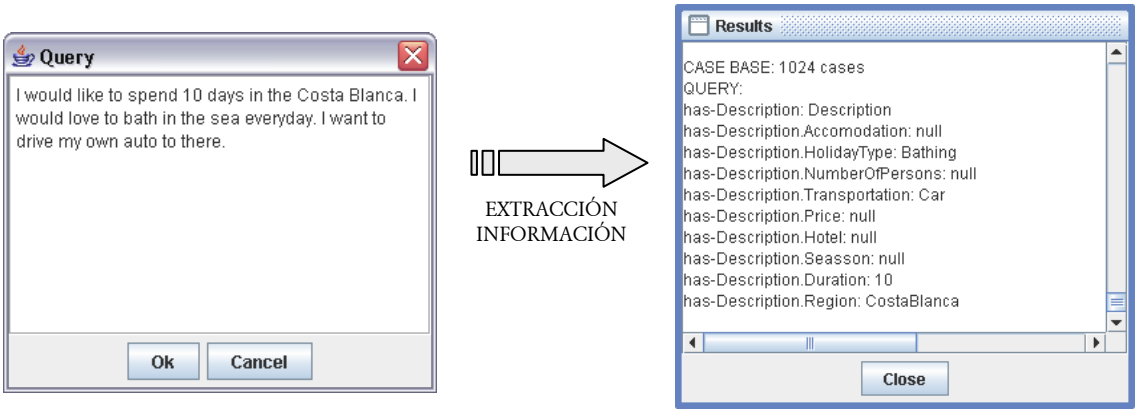
Prueba 2

Consulta

- I would like to spend 10 days in the Costa Blanca. I would love to bath in the sea everyday. I want to drive my own auto to there.

Información extraída

- HolidayType = Bathing
- Transportation = Car
- Duration = 10
- Region = CostaBlanca



En esta segunda prueba, la nueva capa de sinónimos añadida al sistema consigue encontrar en el texto de consulta tres nuevos conceptos. En primer lugar, la etapa de WordNet, al igual que para la prueba anterior, extrae el valor "Bathing" para el atributo "HolidayType" tras haber comprobado que el fragmento del texto "bath" es el infinitivo de la forma verbal "bathing", tal y como se ha explicado en el punto anterior. Para el caso del valor "Car" el comportamiento es similar. En un primer lugar, el

sistema obtiene todas las palabras relevantes de la consulta, entre las que se encuentra “auto”. Después, obtiene la lista de sinónimos de esta palabra, siendo uno de ellos “car”, que coincide con uno de los valores de la enumeración para el atributo “Transportation”, por lo que será extraído como dato relevante.
Para el caso de la etapa de sinónimos definidos por el usuario, se utilizan las siguiente regla para la extracción de información:

```
[Region]{CostaBlanca}([Cc]osta [Bb]lanca)
```

En este caso, esta regla lo único que hace es definir que el concepto “CostaBlanca” puede encontrarse en una consulta con un espacio separando las dos palabras que lo componen, y que éstas pueden no comenzar con mayúscula. De esta forma, la capa de sinónimos definidos por el usuario encuentra dicho concepto y lo almacena como dato relevante.

Fase 3: Confirmación de información extraída

En las dos primeras aproximaciones de nuestro proyecto el sistema analiza las consultas textuales de los usuarios y extrae la información que él considera relevante. Cabe la posibilidad de que dicha extracción de información se haga de forma incorrecta, debido a expresiones propias utilizadas por los usuarios, faltas de ortografía, palabras ambiguas, etc., y el sistema considere ciertos valores erróneos para uno o varios de los atributos que componen la estructura de casos.

Es por esta razón por la que hemos considerado necesario realizar el trabajo de esta tercera aproximación. En ella damos la oportunidad a los usuarios del sistema de verificar los datos que el sistema ha extraído de sus consultas textuales. De esta forma, es posible añadir nuevos valores o, incluso, quitar los que se consideren incorrectos.

La ventaja de permitir la participación del usuario es clara: verificar la información extraída por el sistema y evitar consultas a la base de casos imprecisas. Si algunos datos son erróneos, las funciones de similitud obtendrán valores diferentes que los esperados, y muy seguramente, el resultado no será el esperado o el que en teoría debiera serlo, según las preferencias introducidas por el usuario.

Este trabajo de confirmación de la información extraída es realizado por el sistema en la última etapa de la tarea de obtención de la consulta del usuario ("Obtain query task"), justo después del método de almacenamiento de la información en los atributos de la nueva query estructurada, y justo antes de comenzar a ejecutar las funciones de similitud para cada uno de éstos. Así, el usuario comprueba sus datos antes de lanzar la consulta contra la base de datos, de forma que pueda modificar éstos datos si así lo cree necesario.

Para realizar el trabajo requerido en esta tercera aproximación ha sido necesario plantearse primero un sub-objetivo: tratamiento de los atributos con valores múltiples. El primer problema surgió a la hora de pensar en la representación gráfica de estos valores, ya que el entorno jCOLIBRI no lo permite. También fue necesario pensar en la representación interna (en código Java) que iban a tener estos atributos.

Como ya se ha explicado en las secciones anteriores de este documento, la información contenida en los atributos de una query siempre es textual y en forma de cadena de texto. Si en estos fragmentos se encuentran representados varios valores (por ejemplo, una sucesión de valores de tipos enumerados o de enteros) dicha información estará separada por espacios en blanco.

En esta nueva etapa del sistema la representación anterior ya no es válida, aunque se puede seguir utilizando hasta llegar a ella. Para que esta capa pueda mostrar de forma gráfica los datos al usuario es necesario convertir estos valores a una representación interna determinada, en forma de un tipo propio reconocido por el sistema. En el próximo apartado de este documento se explica en detalle el funcionamiento de los métodos que llevan a cabo dicha transformación.

Las nuevas representaciones de tipos múltiples que hemos llevado a cabo en esta etapa han sido:

- Entero múltiple
- Tipo enumerado múltiple

Con el diseño de estos nuevos tipos, y sus respectivos editores gráficos, hemos proporcionado a nuestro objetivo principal de esta etapa la infraestructura necesaria para desarrollar su trabajo, de forma que pueda mostrar de forma correcta y completa a los usuarios los datos extraídos de sus consultas textuales.

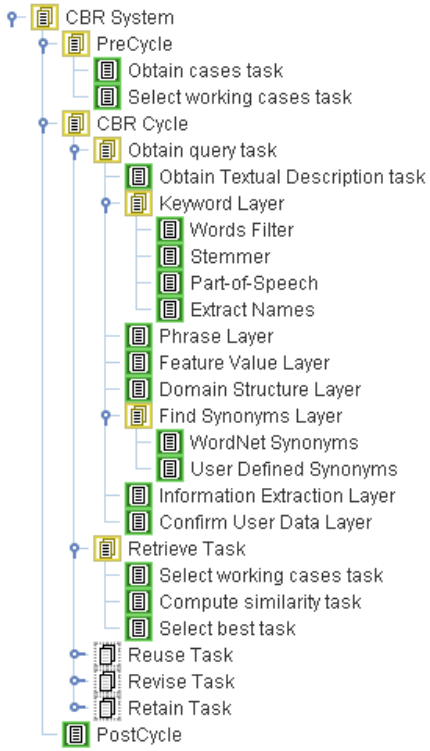
Después de realizar la extracción de información pertinente y crear la nueva query estructurada con ellos, el sistema muestra al usuario un formulario con esos datos, dándole la oportunidad a que los

modifique. Cualquier modificación realizada por éste será actualizada en los atributos de la query contenida en el contexto del sistema CBR para su posterior cálculo de similitudes con cada uno de los casos de la base de casos.

Cómo último detalle cabe destacar que el trabajo realizado en esta tercera etapa del proyecto puede servir como base para una nueva aproximación en que se tengan en cuenta los cambios realizados por el usuario para un posible aprendizaje del sistema CBR, aunque ésto se plantea como utilidad avanzada de la herramienta para el futuro.

Etapas del sistema

En la figura incluida a continuación se puede ver la estructura de nuestro sistema para esta tercera fase de desarrollo, con todas las etapas y tareas que lo componen. Como se puede observar, en esta tercera aproximación del proyecto sólo ha sido necesario añadir una capa más a las que ya formaban parte del sistema CBR. Esta nueva etapa se ha situado en esa posición de forma estratégica, justo después de que se extraiga y almacene la información relevante, y justo antes de que se apliquen las funciones de similitud entre la consulta y los casos y se seleccione el mejor de todos.



A continuación se describen brevemente cada una de las nuevas tareas introducidas, y aquellas que han sido modificadas, en nuestro sistema. En los próximos apartados se explican detalladamente.

- Pre-ciclo (PreCycle). A esta etapa no se han añadido nuevas tareas.
    - Tarea de obtención de casos (Obtain cases task).
    - Tarea de selección de casos activos (Select working cases task).
- **Ciclo CBR** (CBR Cycle).
    - **Tarea de obtención de la query** (Obtain Query Task). En esta etapa ha sido incluida una nueva sub-tarea con la utilidad desarrollada en esta fase del proyecto.
        - Tarea de obtención de la consulta textual (Obtain Textual Description task).
        - Capa de palabras clave (Keyword Layer).
            - Filtro de palabras (Words Filter).
            - Extracción de raíces (Stemmer).
            - Categorías léxicas (Part-of-Speech).
            - Extracción de nombres (Extract Names).

- Capa de expresiones propias (Phrase Layer).
- Capa de características (Feature Value Layer).
- Capa de estructura de dominio (Domain Structure Layer).

- Capa de sinónimos (Synonyms Layer).

    - Sinónimos WordNet (WordNet Synonyms).
    - Sinónimos definidos por el usuario (User Defined Synonyms).

- Capa de extracción de información (Information Extraction Layer).

- **Capa de confirmación de información extraída** (Confirm User Data Layer). La finalidad de esta nueva es mostrar al usuario los datos que han sido recuperados de su consulta textual, de forma que éste tenga la opción de modificarlos (añadiendo o eliminando los que hay) o confirmando sin realizar cambios si así lo considera. Se trata de una capa de resolución, ya que no divide su trabajo en sub-tareas.

o Tarea Recuperar (Retrieve Task).

- Tarea de selección de casos activos (Select working cases task).
- Tarea de cómputo de similitud (Compute similarity task).
- Tarea de selección del mejor (Select best task).

o Tarea Reutilizar (Reuse Task). Esta tarea no se utiliza.
o Tarea Revisar (Revise Task). Esta tarea no se utiliza.
o Tarea Recordar (Retain Task). Esta tarea no se utiliza.

- Post-ciclo (PostCycle). A esta etapa no se han añadido nuevas tareas.

Confirm User Data Layer

Localización: CBR Cycle → Obtain Query Task → Confirm User Data Layer.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.ConfigureConfirmationPane.
Parámetros: ninguno.
Tipo de tarea: resolución.

El método de resolución de esta nueva etapa se encarga de mostrar al usuario mediante un formulario los datos relevantes que han sido extraídos de su consulta textual. Así, éste puede realizar cualquier tipo de modificación para ayudar al sistema a realizar una consulta sobre la base de casos con una mayor precisión.

La primera tarea de esta etapa es obtener la query almacenada en el contexto del sistema y extraer todos sus atributos, que ya estarán rellenos con los valores extraídos en la etapa anterior. La información relativa a cada uno estos atributos que es necesario extraer es la siguiente:

- **Nombre.** Necesario para mostrarlo en el formulario del usuario.
- **Descripción.** Necesario para actualizar el atributo con el nuevo valor modificado.
- **Tipo.** Para seleccionar el editor gráfico adecuado.
- **Valor.** Para mostrar al usuario el valor o valores extraídos anteriormente.

Para construir el formulario que se mostrará al usuario con la información extraída se utiliza la clase provista por jCOLIBRI "GenericMethodParametersPane", a la que es necesario proporcionarle los datos de los atributos anteriores en un formato específico (mediante objetos "MethodParameter"). Este método detecta automáticamente el editor requerido por cada atributo en función de su tipo y construye el formulario paso a paso.

Después de realizar la petición a esta clase de construcción del formulario es necesario rellenar los editores con los valores almacenados en la query del sistema. Para ello, se han de recorrer uno a uno y, haciendo uso del método "setDefaultValue" de las clases que representan a cada uno de los editores, inicializarlos todos con los datos extraídos en las etapas anteriores del sistema CBR.

En esta última acción es donde cada editor debe transformar los datos recibidos (como cadena de texto) en una representación entendible por él. Más adelante se explicará las operaciones específicas diseñadas para los nuevos tipos creados por nosotros.

Después de inicializar los editores hay que mostrar al usuario el formulario con los datos para que éste pueda realizar modificaciones si así lo cree necesario. Por tanto, se ha de esperar a que éste pulse sobre el botón "Aceptar" del formulario para continuar con las acciones de esta etapa, o por el contrario salir de la aplicación si el usuario pulsó sobre el botón "Cancelar".

Por último, si el usuario aceptó el formulario anterior, es necesario actualizar la query estructurada del sistema con los nuevos datos introducidos. Aunque el usuario no realice ningún cambio o sólo los realice en determinados campos, todos los atributos de la query serán sustituidos por los nuevos valores obtenidos del formulario. El único trabajo que hay que realizar en esta etapa es asignar a cada atributo de la query el valor de aquel campo del formulario cuyos nombres coincidan.

A continuación se explican en detalle los nuevos tipos creados con sus correspondientes editores gráficos y las nuevas funciones de similitud local diseñadas para estos nuevos tipos.

MultipleInteger

Clase Java que implementa el tipo: jcolibri.extensions.user.datatypes.MultipleInteger.
Representación de: múltiples valores enteros.

Esta clase representa una secuencia de números enteros, es decir, múltiples valores enteros, y puede ser considerada como una extensión de la clase Integer de Java, que sólo permite la representación de un único valor.

Esta nueva clase contiene una lista en la que almacena todos los enteros que se necesiten.

MultipleIntegerEditor

Clase Java que implementa el editor: jcolibri.extensions.user.gui.editor.MultipleIntegerEditor.
Editor del tipo: MultipleInteger.

Esta clase implementa un editor para el tipo "MultipleInteger", de forma que muestre todos los valores que éste almacena. Para ello, se ha decido utilizar un objeto Java "JTextField" (campo de texto) en el que se enumeran todos estos valores con una separación cada dos de un espacio en blanco, de la forma:

10 15 20

El método "setDefaultValue" de este editor recibe, según se ha comentado en el apartado anterior, una cadena de texto con una sucesión de valores enteros separados por espacio que, al coincidir con la representación gráfica del editor, no es necesario transformar para ser mostrada al usuario.

Por su parte, el método "getEditorValue" de este editor (que devuelve el valor representado) necesita transformar los datos para poder ser retornados. Así, este método obtiene una cadena de texto con la representación explicada antes y la transforma en un objeto del tipo "MultipleInteger".

Para realizar esta transformación es necesario dividir la cadena en los valores que la conforman y crear una lista con ellos, que será la que forma parte del tipo de entero múltiple.

En el ejemplo anterior, se crearía una lista de longitud 3, compuesta por los valores 10, 15 y 20.

La transformación realizada por este editor permite adaptar los datos extraídos por las etapas anteriores del sistema CBR (a modo de fragmentos de texto) en una representación interna entendible por las funciones de similitud que trabajan sobre este tipo de datos de enteros múltiples.

MultipleStringEnum

Clase Java que implementa el tipo: jcolibri.extensions.user.datatypes.MultipleStringEnum.
Representación de: selección múltiple de valores de un tipo enumerado.

Esta clase representa una secuencia de valores de una cierta enumeración, y puede ser considerada como una extensión de la clase "StringEnum" proporcionada por la herramienta jCOLIBRI, que sólo permite la selección de un único valor de ese tipo enumerado.

Esta nueva clase contiene almacena:

- Una enumeración de valores (tipo enumerado).
- Una lista en la que se almacenan todos los valores seleccionados del tipo anterior.

MultipleEnumEditor

Clase Java que implementa el editor: jcolibri.extensions.user.gui.editor.MultipleEnumEditor.
Editor del tipo: MultipleStringEnum.

Esta clase implementa un editor para el nuevo tipo "MultipleStringEnum", de forma que muestre gráficamente todos los valores almacenados por éste. Para ello, se ha decidido utilizar un objeto de tipo "JList", que permite mostrar una serie de valores (al igual que un "JComboBox") pero permite una selección múltiple (a diferencia de la clase "JComboBox").

Al igual que en el editor de enteros múltiples, los datos recibidos son de tipo cadena de texto, con una enumeración de valores separados dos a dos mediante espacios en blanco, de la forma:

June July August

La forma de proceder de este editor es la siguiente. Lo primero que necesita hacer esta clase es, en el método "setConfig", obtener los valores del tipo enumerado sobre el que se va a trabajar (utilidad provista por la clase "DataTypesRegistry") y crear con ellos un objeto de tipo "JList".

Después, en el método "setDefaultValue" el editor ha de obtener los valores que componen el valor del atributo y seleccionarlos en la "JList" anterior, para que el usuario pueda ver que información ha sido extraído en las etapas anteriores del sistema.

En el ejemplo anterior, el editor crearía una "JList" con toda la enumeración de valores del tipo (en este caso concreto, todos los meses del año), extraería de la cadena de texto recibida los tres valores (correspondientes a los meses de Junio, Julio y Agosto) y los seleccionaría en lista anterior.

La selección de este tipo de listas se ha ajustado en este caso a un modo "sencillo", lo que significa que cada uno de los valores que la componen puede ser seleccionado con independencia del resto, es decir, cualquiera de ellos puede ser seleccionado.

Por último, esta clase implementa el método "getEditorValue" que ha de devolver los valores seleccionados de todos los que componen la enumeración. Para realizar esta tarea, el editor crea una lista con los valores seleccionados, que será la que forme parte del tipo "MultipleStringEnum", junto al tipo enumerado utilizado, que se devolverá al llamar a dicha función.

En el ejemplo anterior, se crearía una lista de longitud 3, compuesta por los valores "June", "July" y "August".

Al igual que en el editor de enteros múltiples, éste permite adaptar los datos extraídos por las etapas anteriores en forma de cadena de texto a una representación entendible por las funciones de similitud que posteriormente se aplicarán sobre los atributos con tipo "MultipleStringEnum".

Nuevas funciones de similitud

Además de haber desarrollado en esta aproximación dos nuevos tipos de datos y sus respectivos editores gráficos, ha sido necesario diseñar tres nuevas funciones de similitud local que permitan comparar valores con estos nuevos tipos. Así, se ha implementado una función que compara dos enteros múltiples y otra dos para comparar tipos enumerados múltiples (para distancias cíclicas y no cíclicas).

El funcionamiento de estas nuevas funciones coincide exactamente con el de sus predecesoras. Éstas han sido las sustituciones realizadas:

- La nueva función "**MultipleIntegerInterval**" de similitud entre enteros múltiples sustituye a la anterior función "AverageMultipleTextValues", siendo el comportamiento de ambas el mismo, calcular la media de varios valores enteros y devolver una similitud de 0 a 1 dentro de un intervalo determinado.

- Las nuevas funciones "**MultipleEnumDistance**" y "**MultipleEnumCyclicDistance**" han sido diseñadas para sustituir a las anteriores "MinEnumDistanceMultipleTextValues" y "MinEnumCyclicDistanceMultipleTextValues" respectivamente. El comportamiento de cada función nueva y su predecesora es también el mismo. En el primer caso se calcula la mínima distancia entre todos los valores seleccionados de dos enumeraciones de valores y en el segundo caso igual pero aplicando la función de forma cíclica (el primero y el último valor de la enumeración se encuentran a distancia 1).

El comportamiento detallado de las funciones que han sido sustituidas puede ser consultado en el apartado correspondiente de la primera aproximación del proyecto. A continuación se detalla un pequeño esquema que define el comportamiento de las nuevas funciones de similitud añadidas al entorno jCOLIBRI, ya que, aunque son muy similares a las anteriores, será útil para hacerse una idea general de su funcionamiento.

- **MultipleIntegerInterval.** Esta función de similitud local recibe como parámetros dos valores del nuevo tipo "MultipleInteger" (enteros múltiples) y calcula la media aritmética de todos los valores de cada uno de ellos, por separado. Después calcula la diferencia entre ambos resultados dentro de un intervalo especificado como otro parámetro de la nueva función.

La diferencia con la función a la que ha sustituido es que en la anterior los valores numéricos tenían que ser extraídos de una cadena de texto (valores separados por espacios) y en la actual no es necesario debido al nuevo tipo de datos diseñado en esta aproximación, que provee una operación que retorna todos los enteros que componen el valor múltiple.

Esta nueva función requiere, al igual que la función a la que sustituye, el uso de un parámetro que indique el intervalo a aplicar para obtener la solución, al igual que en el caso anterior.

Ejemplo:

Query	10 18	MediaQ = (10 + 18) / 2 = 14
Caso	10 12 14	MediaC = (10 + 12 + 14) / 3 = 12
Intervalo	20	
Resultado	1 - (| MediaQ - MediaC | / Intervalo) = 0.9	

- **MultipleEnumDistance.** Esta nueva función de similitud local recibe como parámetros dos valores del nuevo tipo "MultipleStringEnum" (enumeración de valores con múltiple selección) y calcula la distancia mínima entre ambos. Para ello, obtiene todos los valores seleccionados de cada uno de los dos parámetros y calcula sus diferentes ordinales (posición del valor dentro de la enumeración). El valor devuelto por la función será la mínima distancia encontrada entre los ordinales del primer parámetro y los del segundo, ponderada por el número total de valores del tipo enumerado.

  La diferencia con la función a la que ha sustituido es que la anterior tenía que extraer los valores seleccionados de una cadena de texto (separados por espacios en blanco) y a partir de ahí calcular la distancia mínima. En este caso, esta operación previa ya no es necesaria debido a la representación del nuevo tipo de datos diseñado en esta aproximación, que provee una función que retorna todos los valores seleccionados de un tipo enumerado, así como todos los valores que componen la enumeración.

  Esta nueva función ya no requiere, a diferencia de la que ha sustituido, un parámetro que le indique la enumeración de valores a utilizar, ya que el nuevo tipo "MultipleStringEnum" ya incluye esta información.

  Ejemplo:

  Tipo Enumerado: Plane (1), Car (2), Train (3), Coach (4)

Query	Plane Car	Ordinales Q = {1, 2}
Caso	Coach	Ordinales C = {4}
Distancias	|1 - 4| = 3 , |2 - 4| = 2 $\rightarrow$ {2, 3}	
Mínima distancia	2	
Resultado	1 - (Min. Dist. / Num. Valores) = 0.5	

- **MultipleEnumCyclicDistance.** Esta nueva función de similitud local se comporta de igual manera que la función anterior, con el único cambio de que en este caso la diferencia entre ordinales que se calcula es cíclica, es decir, la distancia entre el primero y el último elemento de la enumeración de valores es 1, mientras que para la función anterior la diferencia es el número total de valores menos 1.

  La diferencia con la función a la que ha sustituido es la misma que en el caso anterior: los datos se obtienen de forma automática a partir del nuevo tipo "MultipleStringEnum", en vez de tener que ser extraídos de una cadena de texto sin procesar.

Esta nueva función, al igual que la anterior, tampoco requiere ya el paso de un parámetro con los diferentes valores de la enumeración a utilizar.

Ejemplo:

Tipo Enumerado: Plane (1), Car (2), Train (3), Coach (4)

Query	Plane Car	Ordinales Q = {1, 2}
Caso	Coach	Ordinales C = {4}
Distancias no cíclicas	|1 - 4| = 3 , |2 - 4| = 2	→ {2, 3}
Distancias cíclicas	|4 - 3| = 1 , |4 - 2| = 2	→ {1, 2}
Mínima distancia	1	
Resultado	1 - (Min. Dist. / Num. Valores) = 0.75	

Con el desarrollo de los nuevos tipos múltiples y de las tres funciones de similitud local añadidas al entorno jCOLIBRI, la estructura de casos para el dominio específico de los viajes ha quedado de la siguiente forma:

- **HolidayType**
  Tipo: HolidayTypesMultipleEnum
  Función de similitud local: MultipleEnumDistance
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 1.0

- **Price**
  Tipo: MultipleInteger
  Función de similitud local: MultipleIntegerInterval
  Parámetro de la función de similitud: Interval = 2500
  Peso: 1.0

- **NumberOfPersons**
  Tipo: MultipleInteger
  Función de similitud local: MultipleIntegerInterval
  Parámetro de la función de similitud: Interval = 12
  Peso = 0.5

- **Region**
  Tipo: RegionsMultipleEnum
  Función de similitud local: MultipleEnumDistance
  La función de similitud especificada no requiere el uso de parámetros.
  Peso = 1.0

- **Transportation**
  Tipo: TransportationsMultipleEnum
  Función de similitud local: MultipleEnumDistance
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 0.7

- **Duration**
  Tipo: MultipleInteger
  Función de similitud local: MultipleIntegerInterval
  Parámetro de la función de similitud: Interval = 21
  Peso: 0.5

- **Season**
  Tipo: SeasonsMultipleEnum
  Función de similitud local: MultipleEnumCyclicDistance
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 0.7

- **Accommodation**
  Tipo: AccommodationsMultipleEnum
  Función de similitud local: MultipleEnumDistance
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 0.7

- **Hotel**
  Tipo: String
  Función de similitud local: EqualsStringIgnoreCase
  La función de similitud especificada no requiere el uso de parámetros.
  Peso: 0.3

  Función de similitud global: **Average**.

Los tipos "HolidayTypesMultipleEnum", "RegionsMultipleEnum", "TransportationsMultipleEnum", "SeasonsMultipleEnum" y "AccommodationsMultipleEnum" son instancias específicas del nuevo tipo diseñado "MultipleStringEnum". Cada uno de los cinco tiene su propia enumeración de valores que puede tomar el atributo sobre el que se define el tipo.

Por último se detallan los datos específicos de cada una de las nuevas funciones de similitud local diseñadas en esta tercera aproximación del proyecto:

- **MultipleIntegerInterval**
  Clase Java de la función: jcolibri.extensions.user.similarity.local.MultipleIntegerInterval.
  Función de similitud para atributos del tipo: MultipleInteger.
  Parámetros: intervalo de comparación (de tipo entero).

- **MultipleEnumDistance**
  Clase Java de la función: jcolibri.extensions.user.similarity.local.MultipleEnumDistance.
  Función de similitud para atributos del tipo: MultipleStringEnum.
  Parámetros: ninguno.

- **MultipleEnumCyclicDistance**
  Clase Java de la función: jcolibri.extensions.user.similarity.local.MultipleEnumCyclicDistance.
  Función de similitud para atributos del tipo: MultipleStringEnum.
  Parámetros: ninguno.

Nuevas utilidades añadidas

Con las utilidades añadidas en esta tercera aproximación del proyecto se han resuelto dos de las más importantes limitaciones que teníamos desde el comienzo.

Las principales utilidades añadidas en esta etapa son las siguientes:

- Posibilidad de intervención del usuario.
- Representación interna de atributos con valores múltiples.
- Capacidad gráfica para representar atributos con valores múltiples.
- Nuevas funciones de similitud local.

Posibilidad de intervención del usuario

Como es evidente, la principal utilidad añadida al sistema en esta tercera etapa de nuestro proyecto es la posibilidad de que el usuario, además de escribir su consulta textual, pueda verificar los datos extraídos por la aplicación. Según se ha explicado antes, cuando el texto ya ha sido analizado y la información relevante ha sido extraída a los atributos de una query estructurada, una nueva etapa se encarga de generar un formulario de forma dinámica en el que se muestran todos estos valores calculados, de forma que el usuario pueda ver que datos han sido extraídos y pueda modificarlos si así lo cree conveniente.

La principal ventaja de esta nueva funcionalidad es que permite al sistema implicar al usuario en el proceso de obtención de la información que desea, ya que éste puede controlar si los datos extraídos de su consulta son correctos y adecuados, teniendo la opción de modificarlos si así lo desea.

Representación interna de atributos con valores múltiples

Para llevar a cabo el desarrollo de esta nueva etapa ha sido necesario también implementar una nueva representación de tipos con múltiples valores, ya que el entorno jCOLIBRI no proporcionaba esta posibilidad. De esta forma, cuando son varios los valores extraídos para un mismo atributo, es posible que éstos sean representados con los nuevos tipos múltiples diseñados en esta tercera aproximación.

Como ya hemos explicado en el apartado anterior, los nuevos tipos se han diseñado para permitir la representación de enteros múltiples y de enumeraciones de valores con selección múltiple.

Capacidad gráfica para representar atributos con valores múltiples

Además de haber realizado el diseño de dos nuevos tipos múltiples para el entorno jCOLIBRI, ha sido necesario llevar a cabo en esta aproximación el diseño de sus editores gráficos. Estos componentes son necesarios debido a que en la nueva etapa de confirmación de datos se requiere la existencia de estructuras que permitan la correcta visualización de los valores extraídos en las capas anteriores. Para ello, cada tipo ha de tener su editor gráfico correspondiente, para poder mostrar al usuario los datos obtenidos.

Por tanto, en esta etapa ha sido necesario crear dos nuevos editores gráficos: uno para el nuevo tipo de enteros múltiples y otro para el tipo correspondiente a la enumeración de valores. En el caso del primero, se ha optado por utilizar un simple campo de texto en el que se pueden mostrar todos los valores en forma de secuencia separados por espacios en blanco, y para el segundo una estructura desplegable de selección múltiple de valores.

Nuevas funciones de similitud local

Al haber añadido nuevos tipos al entorno jCOLIBRI, también ha sido necesario, además de los editores gráficos correspondientes, realizar el diseño de tres nuevas funciones de similitud local que permitan realizar comparaciones entre valores con dichos tipos. De esta forma, dados dos atributos de tipo entero múltiple o de enumeración de valores múltiple, estas funciones devuelven un resultado numérico de 0 a 1 indicando de menos a más la similitud entre ambos.

Estas tres nuevas funciones de similitud han sustituido a las diseñadas en la primera aproximación del proyecto, que extraían los datos a evaluar de una cadena de texto (valores separados mediante espacios en blanco). Ahora estos valores vienen representados mediante una lista de valores que puede ser obtenida a través de los nuevos tipos diseñados de enteros múltiples y enumeración de valores con múltiple selección. El comportamiento de estas nuevas funciones y el de sus antecesoras es exactamente el mismo, con la única diferencia en la obtención de los valores a evaluar y, por tanto, de los tipos para los que han sido diseñadas.

Limitaciones del sistema

En esta tercera etapa de nuestro proyecto hemos conseguido reducir el número de limitaciones de nuestro sistema (siempre dentro de nuestros objetivos). En este caso, las posibles limitaciones que podrían aparecer en sistema vendrían en un futuro en el que se necesitara una mayor flexibilidad o capacidad de representación de los datos con los que se trabaja. Estas limitaciones son:

- Representación interna de la información extraída del texto de consulta.
- Limitación en la representación gráfica para el nuevo tipo de enteros múltiples.

Algunas de las limitaciones que aún permanecen en el sistema son:

- El sistema no permite la representación de negaciones, conjunciones y disyunciones.
- El sistema no es capaz de obtener información relativa a un determinado concepto.

Representación interna de la información extraída del texto de consulta

El entorno jCOLIBRI sólo permite una representación muy sencilla de los datos extraídos del texto de consulta de los usuarios. Como ya hemos explicado anteriormente, esta representación consiste en cadenas de texto que incluyen dichos valores separados dos a dos mediante espacios en blanco.

Debido a esta seria limitación de entorno utilizado, las utilidades desarrolladas por nosotros se han de adaptar a dicha representación. Los nuevos tipos diseñados tienen que extraer los datos de esta forma para poder ser adaptados. Además, así se impide poder añadir más información como, por ejemplo, sobre negaciones, conjunciones o disyunciones encontradas en los textos de consulta de los usuarios.

Por tanto, sería necesario ampliar la capacidad de representación interna de datos extraídos del entorno jCOLIBRI para dotar a los sistemas diseñados de una mayor flexibilidad.

Limitación en la representación gráfica para el nuevo tipo de enteros múltiples

El editor gráfico correspondiente al nuevo tipo de enteros múltiples, diseñado por nosotros en esta etapa del proyecto, presenta una serie limitación en su representación gráfica de los datos. Éste está compuesto de un sencillo campo de texto en el que se enumerar todos los valores numéricos del tipo separados por espacios en blanco. Esta representación es bastante sencilla, aunque también funcional y práctica, pero presenta la limitación de no ofrecer claridad en la presentación de resultados a los usuarios del sistema.

Lo ideal sería haber diseñado un editor gráfico con una mayor capacidad, que representara los valores mediante el uso de componentes independientes para cada dígito. Esta opción fue descartada debido al tiempo que deberíamos haber invertido en ella, ya que de la forma que lo hemos hecho la funcionalidad es prácticamente la misma.

Problemas encontrados

En esta aproximación hemos conseguido, a partir de un problema grande que nos surgió al comienzo, crear una nueva utilidad para el sistema, al mismo tiempo que lo resolvíamos. Se trata de la capacidad añadida de representación de atributos con valores múltiples, así como la expresión gráfica de éstos.

Los principales problemas encontrados a lo largo de esta tercera aproximación han sido:

- Imposibilidad de representación de atributos con valores múltiples.
- Dificultad de adaptación de algunas clases de jCOLIBRI a nuestro código.
- Inestabilidad de la herramienta al modificar código general.

Imposibilidad de representación de atributos con valores múltiples

Como ya hemos explicado en apartados anteriores, el trabajo principal a realizar en esta aproximación se centraba en crear una interfaz gráfica en la que los usuarios de la aplicación pudieran confirmar los datos extraídos de las consultas realizadas por éstos. Este componente gráfico representa, a modo de formulario o plantilla, todos los valores encontrados para cada uno de los atributos que forman parte de la estructura de casos del dominio específico del sistema CBR. Si sólo se encuentra un valor (o ninguno) para cada atributo, el entorno jCOLIBRI no proporciona ningún problema. Pero si el número asciende (al menos dos) nos encontramos con un problema de representación: los tipos diseñados y provistos por jCOLIBRI no permiten la representación de valores múltiples.

Esta representación de valores múltiples, según se ha explicado ya antes, puede ser "simulada" mediante una representación consistente en una cadena de texto con los diferentes valores separados mediante espacios en blanco, aunque esta aproximación no permite apenas flexibilidad. Es por ésto por lo que en esta etapa del proyecto se han diseñado dos nuevos tipos para el entorno que permite representación múltiple de valores, tanto para enteros como para enumeraciones. Cabe destacar que ésta también era una de las limitaciones que presentaba el sistema en su primera aproximación.

Una vez creados estos dos nuevos tipos ya se pudo llevar a cabo el trabajo de implementación de la interfaz gráfica de confirmación de datos por parte de los usuarios de la aplicación.

Dificultad de adaptación de algunas clases de jCOLIBRI a nuestro código

Con el fin de adaptarnos de forma totalmente homogénea al código del entorno jCOLIBRI, a lo largo de todo el proyecto se ha intentado reutilizar lo máximo posible las clases Java ya existentes, aunque ésto nos ha supuesto problemas en diversas ocasiones.

Uno de estos casos fue la reutilización de clase provista por jCOLIBRI "GenericMethodParameters Pane", que permite construir un formulario genérico de forma dinámica, pasándole para ello los atributos y sus correspondientes valores. Aunque en realidad no supuso un problema, si fue una dificultad el adaptarla al código desarrollado por nosotros en esta etapa del proyecto, debido a que hizo falta realizar un mayor procesamiento de información en algunos casos para obtener los datos específicos que requiere esta clase para su utilización.

Inestabilidad de la herramienta al modificar código general

Este problema surgió al intentar adaptar el código de nuestro nuevo método de confirmación de datos con la clase "GenericMethodParametersPane" provista por jCOLIBRI. Para conseguir una correcta interacción entre ambos componentes se modificó una línea de código del método genérico provisto

por el entorno, lo que hizo que todo dejara de funcionar sin ser nosotros conscientes de que había sido ese cambio lo que había provocado tal desastre.

Tras mucho investigar descubrimos finalmente que ese pequeño cambio había sido el causante de ese funcionamiento erróneo y lo modificamos de nuevo a su estado original, para pasar a modificar el nuevo método que estábamos diseñado con el fin de adaptarlo sin tener que realizar ningún cambio en la clase proporcionada por jCOLIBRI, con el consecuente incremento de trabajo.

Pruebas realizadas

En este último apartado de esta fase se detallan algunas de las pruebas que se han realizado en el sistema para comprobar su correcto comportamiento. En este caso se va a destacar el funcionamiento de la nueva utilidad desarrollada: confirmación de información extraída. Como ya hemos explicado antes, después de analizar la consulta de un usuario y extraer los datos relevantes que la componen, el sistema muestra un formulario con dichos datos, permitiendo que éstos puedan ser modificados o simplemente confirmados si la recuperación ha sido exacta.

En la consulta siguiente se destacan los datos relevantes mediante un subrayado, con el fin de facilitar su identificación.

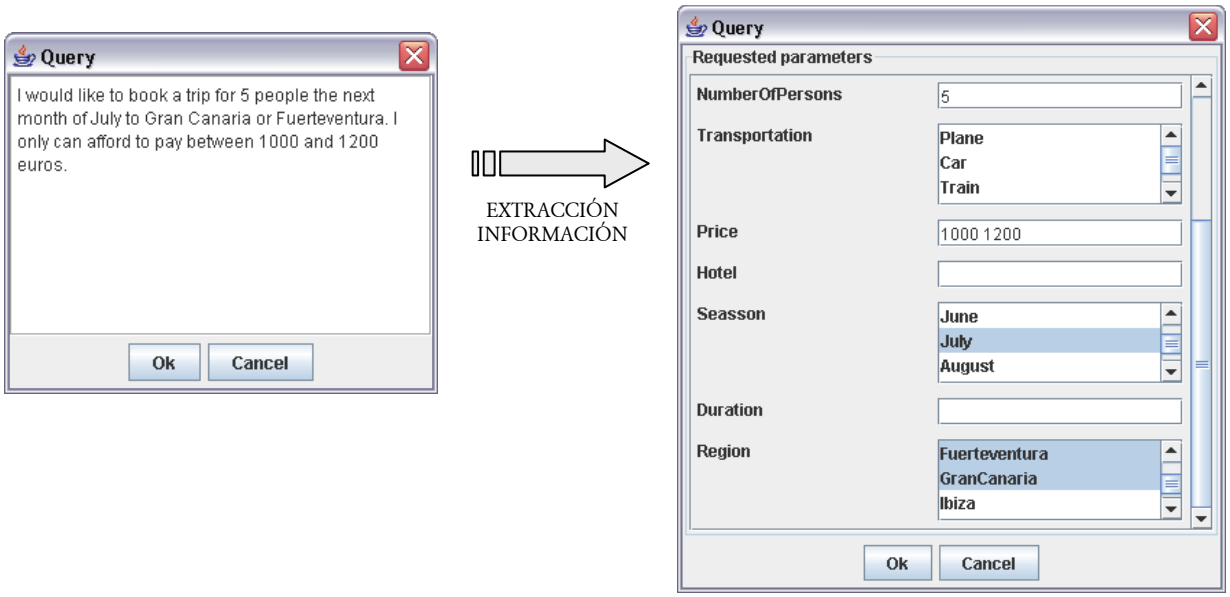
Prueba 1

Consulta

- I would like to book a trip for 5 people the next month of July to Gran Canaria or Fuerteventura. I only can afford to pay between 1000 and 1200 euros.

Información extraída

- NumberOfPersons = 5
- Price = 1000 1200
- Season = July
- Region = Fuerteventura GranCanaria



En esta primera prueba el sistema utiliza la capa "Feature Value Layer" para extraer del texto de consulta del usuario todos los datos relevantes que ésta contiene. En dicha etapa se utiliza una serie de reglas definidas en un fichero de configuración externo creado por el desarrollador, en el que se describen patrones de reconocimiento de características y expresiones particulares del lenguaje. De esta forma, el sistema es capaz de extraer los 6 conceptos importantes que contiene la consulta. Al llegar a la nueva

capa desarrollada en esta fase, “Confirm User Data Layer”, se recuperan éstos datos y se utiliza un formulario genérico que permite mostrárselos al usuario.

Los atributos de la estructura que son de tipo enumerado (“MultipleStringEnum”) se visualizan en una lista que contiene todos los valores de la enumeración con los encontrados en la consulta seleccionados. Los parámetros de tipo entero múltiple (“MultipleInteger”) se muestran en un campo de texto con todos los valores que forman parte separados dos a dos mediante espacios en blanco. Para los atributos que no contengan ningún valor se utilizan editores en blanco.

Una vez que el usuario visualiza sus datos tiene la opción de modificarlos, añadiendo o eliminando, o simplemente aceptarlos para continuar con el proceso de búsqueda de respuesta.

La principal ventaja que presenta la visualización implementada en esta fase es que permite representar todas las estructuras de casos posibles que pueda tener nuestro sistema CBR, con cualquier número de atributos y sean del tipo que sean. Si añadimos un nuevo tipo de datos al entorno jCOLIBRI, sólo es necesario incluir una clase Java que permita representarlo, de forma que se permita la comunicación entre el sistema y sus usuarios.

Fase 4: Utilización de ontologías

Con la inclusión de las ontologías en nuestro trabajo se ha podido llevar a cabo un gran avance a la hora de organizar y tratar todo el conocimiento disponible sobre el dominio en el que estamos trabajando (en este caso concreto el de los viajes, aunque siempre puede ser aplicado a otro cualquiera).

Como ya se ha explicado en la primera parte de este documento, las ontologías son unas estructuras que nos permiten representar el conocimiento de un cierto dominio mediante el uso de clases (o conceptos), ejemplares (o instancias) y relaciones entre éstos (propiedades). De esta forma, podemos expresar todo lo que se conoce sobre un cierto dominio de una forma ordenada y jerarquizada, para después poder consultar estos datos en busca de información similar.

En esta cuarta aproximación del proyecto se ha necesitado crear una nueva etapa al ciclo CBR de nuestro sistema, añadiéndola a las que ya teníamos antes. Como siempre, se ha implementado un método Java que es el que realiza todo el trabajo y, por tanto, implementa la tarea.

El trabajo realizado en esta etapa se ha dividido en dos partes principales:

- Desarrollo de una ontología sobre viajes.
- Implementación de una clase que interactúa con la ontología y extrae información de ella.

Desarrollo de una ontología sobre viajes

Para llevar a cabo la tarea de crear una ontología sobre viajes (dominio sobre el que trabajamos) ha sido necesario utilizar la herramienta de código abierto **Protégé**, desarrollada por la universidad californiana de Stanford. Esta plataforma proporciona una gran cantidad de estructuras de modelado del conocimiento y acciones que permiten la creación, visualización y manipulación de ontologías en varios formatos de representación.

Protégé proporciona dos importantes alternativas para modelar ontologías:

- El editor **Protégé-Frames** permite desarrollar ontologías basadas en marcos (representación de relaciones y propiedades de conceptos mediante "slots"), de acuerdo con el protocolo OKBC (Open Knowledge Base Conectivity).

- El editor **Protégé-OWL** permite desarrollar ontologías para la Web Semántica, en particular en el lenguaje OWL (Web Ontology Language) del W3C (World Wide Web Consortium). Una ontología OWL puede incluir descripciones de clases, propiedades e instancias. A partir de este tipo de estructura, la semántica forma de OWL permite especificar como obtener "consecuencias lógicas", es decir, hechos que no se representan explícitamente pero que pueden ser inferidos.

Para nuestro proyecto hemos optado por crear una ontología de tipo **OWL**, en parte por ser éste el lenguaje de definición de ontologías estándar que ha sido desarrollado más recientemente. Además, OWL tiene como objetivo facilitar un modelo de marcado construido sobre RDF, codificándolo en XML.

Es necesario precisar que OWL consta de tres variantes o sub-lenguajes:

- **OWL-Lite.** Es el más sencillo de los tres, sintácticamente hablando. Suele ser utilizado en ocasiones en las que sólo se requiere una simple jerarquía de clases con sus posibles propiedades y restricciones.

- **OWL-DL.** Esta variante es mucho más expresiva que OWL-Lite y está basada en la lógica de descripción (Description Logics, de aquí el nombre DL).

- **OWL-Full.** Es el más expresivo de los tres sub-lenguajes Se utiliza en situaciones donde una alta expresividad es mucho más importante que la posibilidad de garantizar la completitud computacional del lenguaje. Por tanto, no es posible llevar a cabo un razonamiento automático en las ontologías OWL-Full.

En nuestro caso, ha sido suficiente utilizar la variante OWL-Lite, ya que sólo se requiere la definición de una taxonomía de clases, además de una serie de propiedades que relacionan los ejemplares que forman parte de la ontología.

Una vez hechas las elecciones pertinentes sobre el desarrollo y la representación de la ontologías, es necesario definir la estructura que ésta ha de tener. Cómo ya hemos visto, una ontología es una jerarquía de conceptos, compuesta por ejemplares de éstos y con relaciones entre ellos. Por tanto, el primer paso es definir esta taxonomía de clases.

En cualquier dominio sobre el que se trabaje, los primeros conceptos en definirse tienen que ser los correspondientes a las "clases raíz" (Hierarchy Root Classes), es decir, los conceptos padre de la ontología o los más generales, sobre los que se crearán los demás. Estas clases pueden ser obtenidas analizando el dominio y seleccionando los atributos o propiedades presentes en todos los casos.

Para nuestro dominio particular de los viajes, estas clases raíz coinciden con la división que ya se realizó para crear la base de casos, es decir, coinciden con la estructura de casos. Es necesario advertir que en esta ocasión sólo tienen sentido los atributos que representan conceptos, como puede ser un destino o un hotel, por lo que los atributos con valores numéricos serán suprimidos. Ésto es debido a que es imposible representar de forma genérica un número, ya que puede tomar infinitos valores.

Los conceptos escogidos como clases padres para nuestra ontología han sido los siguientes:

- **Alojamiento** (Accommodation). Clase que engloba a todos los lugares de alojamiento existentes, como hoteles, hostales, paradores, etc.

- **Categoría de alojamiento** (AccommodationRating). Clase que engloba todas las categorías de los alojamientos anteriores, por ejemplo, el número de estrellas de un hotel.

- **Actividad** (Activity). Clase que engloba todas las actividades que se pueden realizar en un cierto destino, como ir a la playa o visitar museos.

- **Destino** (Destination). Clase que engloba a todos los destino turísticos de nuestra base de conocimiento, por ejemplo, Roma, Berlín o Fuerteventura.

- **Temporada** (Season). Clase que engloba a todas las épocas del año en que es posible realizar un viaje, como por ejemplo cada uno de los meses del año o, de forma más general, por estaciones.

- **Transporte** (Transport). Clase que engloba a todos los medios de transporte con los cuáles es posible realizar los viajes, como avión o barco.

A partir de esta primera división de conceptos se desarrolla el resto de la ontología. Ahora es necesario especializar estas primeras clases en sub-conceptos, de forma que se establezca la jerarquía que queramos. En este punto es necesario decidir la estructura que tendrá nuestro ontología, es decir, si necesitamos que sea muy amplia y con muchas clases o algo más sencillo. Cuanto más completa necesitemos que sea, mayor número de clases tendrá, siendo éstas más específicas. Con un ejemplo se puede ver mejor a qué nos referimos.

Consideremos el concepto o clase "Destino". Podemos definir una primera estructura de sub-conceptos con las siguientes clases y ejemplares:

- África
  - Egipto
  - Marruecos
  - Túnez
- América
  - Canadá
  - Estados Unidos
  - México
- Asia
  - China
  - India
  - Japón
- Europa
  - España
  - Francia
  - Italia
- Oceanía
  - Australia
  - Nueva Zelanda

En este caso, se ha realizado una división muy sencilla, en la que los continentes son clases de segundo nivel (por debajo de la clase padre Destino) y los países instancias. Esta división nos puede ser útil siempre que no necesitemos más información, por ejemplo, sobre ciudades.

Ahora consideremos la siguiente jerarquía:

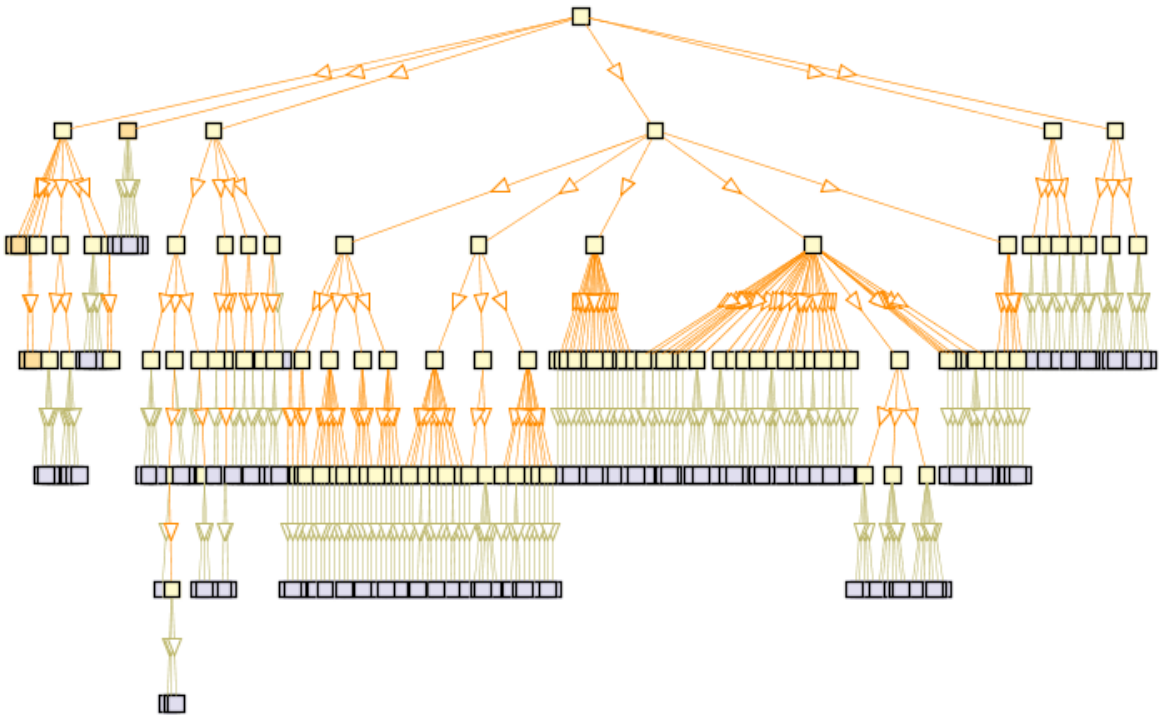
- África
  - Norte de África
  - Sur de África
- América
  - Norte de América
  - Sur de América
- Asia
  - Norte de Asia
  - Sur de Asia
- Europa
  - Norte de Europa
    - Finlandia
    - Noruega
    - Suecia
  - Sur de Europa
    - España
      - Madrid

        - Barcelona
        - Sevilla
        - Fuerteventura
    - Grecia
    - Italia
- Oceanía
    - Norte de Oceanía
    - Sur de Oceanía

En este segundo caso, la información representada es mucho mayor. Para cada instancia, que en este caso se corresponden con las ciudades, sabemos a qué país pertenece, a qué continente y si es del norte o del sur de este.

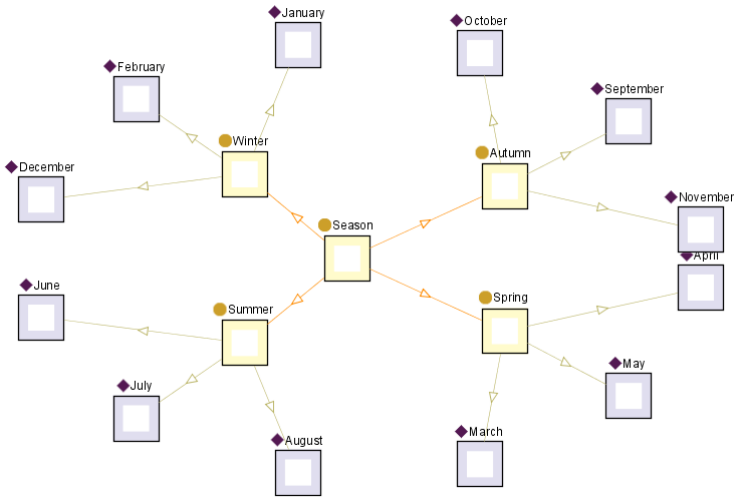
La conclusión que se obtiene de todo esto es que es necesario decidir previamente la información que necesitamos representar en nuestra ontología, para a partir de ahí obtener una división adecuada de sub-clases e instancias. En los casos anteriores, si nuestros viajes tienen como destino ciudades, será necesario incluir esta información, pero si sólo indican el país de destino no será necesario incluir estos datos, convirtiendo entonces los países en los ejemplares de la ontología.

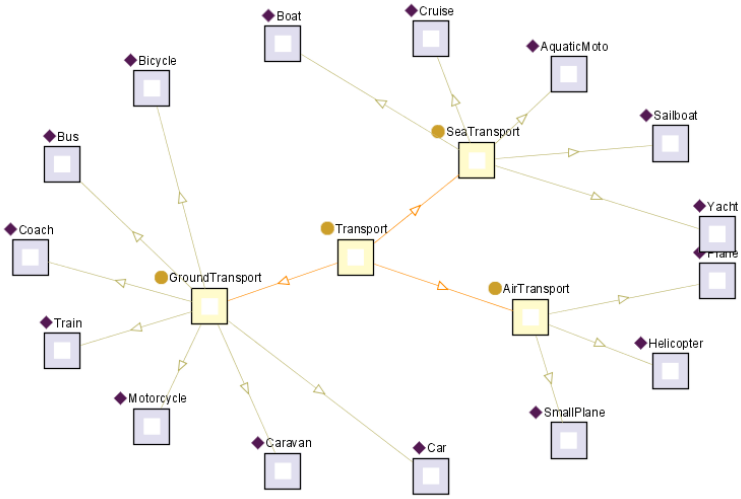
La siguiente figura muestra una visión general de la ontología que hemos desarrollado en esta fase del proyecto. Aunque en ella es difícil distinguir los datos, es útil para poder apreciar su magnitud. En esta figura, los cuadrados representan clases o ejemplares, estos últimos situados en las hojas del árbol, y las líneas relaciones (o propiedades) entre éstos.



Con la ontología que hemos diseñado se ha pretendido representar de forma sencilla parte del extenso conocimiento que existe en el dominio de los viajes, de forma que nos permita extraer información relativa a la base de casos del sistema CBR. En el Apéndice A de este documento se puede consultar la ontología completa.

En las siguientes figuras se muestran algunas de las clases y ejemplares de nuestra ontología en detalle, para los conceptos de "Season" (época) y "Transport" (transporte):





Además de las clases y las instancias de la ontología, ha sido necesario definir una serie de propiedades que relacionen unos ejemplares con otros. Estas propiedades son las siguientes:

- **hasActivity** (tieneActividad)
  Dominio: Destination
  Rango: Activity
  Significado: qué actividad o actividades se pueden realizar en un determinado destino.

- **hasAccommodation** (tieneAlojamiento)
  Dominio: Destination
  Rango: Accommodation
  Inversa de: isLocatedAt
  Significado: qué alojamiento o alojamientos dispone un cierto destino.

- **hasRating** (tieneCategoría)
  Dominio: Accommodation
  Rango: AccommodationRating
  Significado: qué categoría tiene un determinado alojamiento.

- **isLocatedAt** (seEncuentraEn)
  Dominio: Accommodation
  Rango: Destination
  Inversa de: hasAccommodation
  Significado: dónde está localizado un determinado alojamiento (en qué destino se encuentra).

El dominio de una propiedad indica qué ejemplares de la ontología la tienen y el rango los valores que ésta puede tomar. Por ejemplo, para la propiedad "hasAccommodation", el dominio indica que los ejemplares de la clase Destination la tienen, es decir, todos los destinos tienen la propiedad de tener alojamiento; y el rango indica que las instancias de la clase Accommodation son los posibles valores.

Es necesario notar que existen dos propiedades inversas: "hasAccommodation" y "isLocatedAt". Es evidente que si un determinado alojamiento se encuentra en un destino concreto, ese destino dispone de ese alojamiento.

Con la definición de estas propiedades, la ontología nos va a permitir inferir cierto conocimiento que no se representa explícitamente, por ejemplo, todos los alojamiento que tienen una cierta categoría (cuatro estrellas, cinco estrellas...) o en qué destinos se puede realizar una cierta actividad (como visitar museos o darse un baño en la playa).

En el Apéndice A de esta memoria también se pueden consultar estas propiedades definidas y los valores que toman para cada ejemplar.

A continuación se señalan algunos detalles particulares de nuestra ontología:

- La clase "AccommodationRating" es una **clase definida**, similar a una enumeración de valores. En este caso, se define una propiedad "necesaria y suficiente" para ser miembro de esta clase y que obliga simplemente a tomar un valor entre un conjunto de varios permitidos (FiveStars, FourStars...).

- Todas las instancias de las clases concepto "AverageAccommodation", "BudgetAccommodation" y "FirstClassAccommodation" y sus respectivas sub-clases, son ejemplares inferidos. Estas clases definen una serie de restricciones para poder pertenecer a ellas, relativas a la categoría de alojamiento:

  - **BudgetAccommodation**. A esta clase sólo pueden pertenecer los alojamientos que tengan como valor en su propiedad "hasRating" el valor "OneStar" o "TwoStars" de la clase AccomodationRating.

  - **AverageAccommodation**. A esta clase sólo pueden pertenecer los alojamientos que tengan como valor en su propiedad "hasRating" el valor "ThreeStars" o "HolidayFlat" de la clase AccomodationRating.

  - **FirstClassAccommodation**. A esta clase sólo pueden pertenecer los alojamientos que tengan como valor en su propiedad "hasRating" el valor "FourStars" o "FiveStars" de la clase AccomodationRating.

  Por tanto, la clasificación de los ejemplares de la clase "Accommodation" se realizará de forma automática (inferida) bajo estas clases, dependiendo de su categoría. Es necesario notar que estas instancias ya se definieron previamente bajo otros conceptos, correspondientes a los diferentes tipos de alojamiento (hotel, hostal, parador...). Estas clases son también clases definidas.

Implementación de una clase que interactúa con la ontología y extrae información de ella

La segunda gran parte del trabajo realizado en esta etapa del proyecto se la lleva el desarrollo de una clase Java encarga de interactuar con la ontología anterior y obtener información a partir del conocimiento representado en ella. Aunque en el próximo apartado se explicará detalladamente su funcionamiento, aquí se van a introducir algunas líneas general de cómo se comporta.
La principal ventaja que nos proporciona la utilización de una ontología es que nos permite obtener cierto conocimiento relativo a un determinado concepto. Por ejemplo, considerando el destino "Fuerteventura" y observando su clasificación en la ontología, podemos determinar la siguiente información:

- Se encuentra en las Islas Canarias, que a su vez están en España y a su vez en Europa.
- En este destino se pueden realizar las actividades tomar el sol (Sunbathing) y bañarse (Bathing).
- Tiene como alojamientos los hoteles "SolGorriones" e "IberostarPlayaGaviotas".

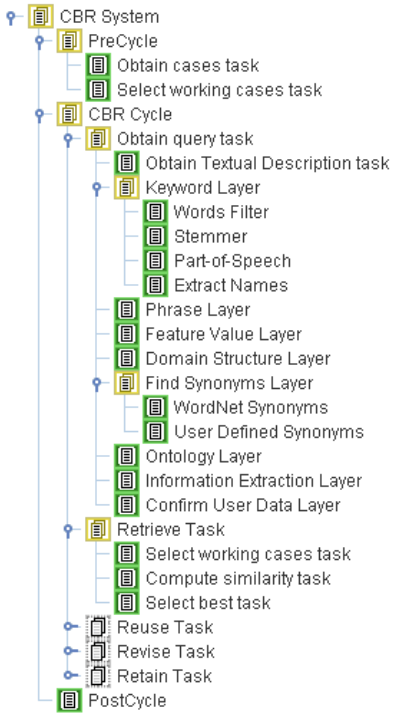
Ahora consideremos que Fuerteventura no es un destino posible entre los que se ofertan (lo que es equivalente a decir que este concepto no aparece en la base de casos de nuestro sistema CBR) pero que el usuario ha especificado que quiere que este sea el destino de su viaje. De esta forma, el sistema puede tomar la sencilla solución de no devolver ningún resultado, ya que existe ningún viaje a Fuerteventura, pero lo ideal sería que obtuviera destinos similares para ofrecérselos al usuario en vez del destino que él propone.

Este conocimiento relativo o similar puede ser obtenido a partir de la ontología que hemos diseñado. Así, primero se extraerían las propiedades definidas para Fuerteventura (vistas antes) y se buscarían destinos con las mismas o lo más parecidas posibles, es decir, lugares donde también se pueda tomar el sol y bañarse y que, preferiblemente, se encuentren en las Islas Canarias, como pueden ser Tenerife, Lanzarote o Gran Canaria.

De esta forma, el sistema ofrecería al usuario viajes con estos destinos y que probablemente satisfarán las necesidades de éste al ser muy similares a sus preferencias.

Etapas del sistema

En la figura que se presenta a continuación se detallan cada una de las etapas en las que se divide nuestro sistema en esta cuarta fase de su desarrollo. En este caso sólo ha sido necesario añadir una nueva etapa, que se corresponde con la tarea que interactúa con la ontología diseñada previamente. Esta capa se ha situado justo antes de la de extracción de información (Information Extraction Layer) y justo después de las de análisis de la consulta textual del usuario. De esta forma, las etapas previas obtienen los conceptos que representan la información referida por los usuarios, que puede ser complementada o sustituida por la nueva capa de ontologías. Por último, y como venía siendo, se extraen estos datos para crear la consulta que se ha de realizar sobre la base de casos.



A continuación se explica brevemente esta nueva etapa introducida en nuestro sistema. En los siguientes apartados se detalla su comportamiento de forma precisa.

- Pre-ciclo (PreCycle). A esta etapa no se han añadido nuevas tareas.
    - Tarea de obtención de casos (Obtain cases task).
    - Tarea de selección de casos activos (Select working cases task).
- **Ciclo CBR** (CBR Cycle).
    - **Tarea de obtención de la query** (Obtain Query Task). En esta etapa ha sido incluida una nueva sub-tarea con la utilidad desarrollada en esta fase del proyecto.
        - Tarea de obtención de la consulta textual (Obtain Textual Description task).
        - Capa de palabras clave (Keyword Layer).
            - Filtro de palabras (Words Filter).
            - Extracción de raíces (Stemmer).

        - Categorías léxicas (Part-of-Speech).
        - Extracción de nombres (Extract Names).

    - Capa de expresiones propias (Phrase Layer).
    - Capa de características (Feature Value Layer).
    - Capa de estructura de dominio (Domain Structure Layer).

    - Capa de sinónimos (Synonyms Layer).

        - Sinónimos WordNet (WordNet Synonyms).
        - Sinónimos definidos por el usuario (User Defined Synonyms).

    - **Capa de Ontologías** (Ontology Layer). Esta nueva etapa del sistema intenta encontrar información relacionada con los datos extraídos de las consultas de los usuarios mediante su interacción con una ontología, sobre el mismo dominio sobre el que trabaja el CBR. De esta forma, es posible obtener conceptos relativos o similares que complementen el proceso de análisis de los textos.

    - Capa de extracción de información (Information Extraction Layer).
    - Capa de confirmación de información extraída (Confirm User Data Layer).

  - Tarea Recuperar (Retrieve Task).

    - Tarea de selección de casos activos (Select working cases task).
    - Tarea de cómputo de similitud (Compute similarity task).
    - Tarea de selección del mejor (Select best task).

  - Tarea Reutilizar (Reuse Task). Esta tarea no se utiliza.
  - Tarea Revisar (Revise Task). Esta tarea no se utiliza.
  - Tarea Recordar (Retain Task). Esta tarea no se utiliza.

- Post-ciclo (PostCycle). A esta etapa no se han añadido nuevas tareas.

Ontology Layer

Localización: CBR Cycle → Obtain Query Task → Ontology Layer.
Clase Java que implementa la tarea: jcolibri.extensions.user.method.OntologyLayerMethod.
Parámetros: fichero OWL donde se encuentra definida la ontología a utilizar (de tipo File).
Tipo de tarea: resolución.

Para llevar a cabo la implementación de este método primero se pensó en un algoritmo que nos proporcionara el comportamiento deseado de extracción de información relativa a un cierto concepto que antes habíamos planteado. En las próximas líneas se explica detalladamente el comportamiento de este algoritmo.

Lo primero que es necesario hacer es extraer de la consulta del usuario los conceptos identificados en las etapas anteriores, y que han sido almacenados en los atributos FEATURES de cada párrafo que forma parte de este texto. Recordemos que estos conceptos son obtenidos mediante el análisis sintáctico de las etapas previas del sistema CBR, aplicando una serie de reglas que permiten identificar características y expresiones propias del lenguaje.

Una vez extraídos todos estos conceptos, que pueden ser considerados como representantes de las preferencias del usuario, es necesario comprobar, para cada uno ellos, si existen en la base de casos, es decir, si algún caso de la base incluye este concepto. Por ejemplo, para los destinos de los viajes, si se ha extraído de la consulta del usuario el concepto Berlín, será necesario comprobar si alguno de todos los viajes almacenados en la base de casos tiene este lugar como destino. En el caso de que este valor exista y se encuentre, el concepto se mantendrá como parte de la consulta del usuario, ya que él será lo más parecido a si mismo, es decir, si un concepto coincide exactamente con otro de la base de casos no será necesario obtener información similar sobre la ontología ya que se dispone de información exacta.

Es en el caso de no haber coincidencia exacta es cuando se recurrirá a la ontología para extraer información relativa o similar. De esta forma, este concepto obtenido de la consulta textual del usuario será eliminado de la lista de conceptos encontrados, para pasar a añadir nuevos valores obtenidos a partir de la interacción con la ontología. La eliminación de este concepto encontrado se debe a que tiene similitud cero con todos los casos de la base de casos, al no coincidir de forma exacta con ninguno de los valores que éstos almacenan.

Para cada uno de los conceptos que no se encuentren en la base de casos se buscará información similar en la ontología sobre la que se va a trabajar. Para ello, el primer paso es localizar el concepto en la taxonomía. En el caso de que no se encuentre se continuará con el siguiente. Si se encuentra será necesario determinar si se trata de una clase o de una instancia, ya que el comportamiento a seguir a continuación es diferente en cada uno de los dos casos. Es importante notar en este punto la necesidad de convergencia entre los nombres utilizados para los conceptos en la ontología y los utilizados para los conceptos en la base de casos, de forma que éstos coincidan si se refieren a lo mismo.

En el caso de que el concepto buscado sea una clase la forma de proceder es la siguiente:

- El primer paso es obtener todas las sub-clases de esta clase. Si se encuentran resultados, es decir, si existen sub-clases, comprobar si existen en la base de casos del sistema. En caso de que sí existan se añadirán a la lista de características extraídas que más tarde se almacenarán en los atributos estructurados de la query.

- En el caso de que el paso anterior no se haya añadido ninguna nueva característica, se pasará a buscar todas las instancias de la clase (instancias a su vez de sus sub-clases). Como en el caso anterior, se comprobará si estos conceptos existen en la base de casos, almacenados si la respuesta es afirmativa.

Para explicar este funcionamiento podemos recurrir al siguiente ejemplo. Consideremos que el usuario desea viajar este verano. El sistema analizaría la consulta y extraería el concepto "Verano" ("Summer"). Al mirar en la base de casos comprobaría que éste no existe, ya que las épocas de viajes están representadas por los meses del año. Por tanto, buscaría el concepto en la ontología y descubriría que se trata de una clase. Al no tener esta clase sub-clases, el algoritmo buscaría todas las instancias de "Verano", siendo éstas los ejemplares "Junio", "Julio" y "Agosto", y comprobaría que efectivamente existen en la base de casos. De esta forma, almacenaría como conceptos extraído estos tres valores. Este comportamiento lo denotaremos con el nombre de "especialización".

En el caso de que el concepto buscado en la ontología no sea una clase, sino una instancia, las acciones a realizar serían las siguientes:

- El primer paso es extraer todas las propiedades del ejemplar, incluyendo aquí las referentes a su situación en la taxonomía, es decir, todas las clases que son padre de este concepto, también conocidas como superclases. Todas estas propiedades serán agrupadas por tipo para facilitar su próximo procesamiento.

- El siguiente paso es eliminar de todas las propiedades anteriores las que sean "exclusivas", es decir, las que sólo este ejemplar tiene entre todas las instancias de la ontología. Ejemplos de este tipo de propiedades son "hasAccommodation" y "isLocatedAt" para los ejemplares de la clase "Destination" y "Accommodation" respectivamente. Es evidente que un alojamiento sólo puede estar situado en un determinado destino y viceversa.
- La siguiente tarea a realizar es obtener, a partir de todas las superclases encontradas antes, seleccionar aquellas que sean padres directos del ejemplar con el que estamos trabajando, es decir, los conceptos que se encuentren justamente un nivel por encima de donde se encuentra la instancia.

- El siguiente paso es buscar nuevas instancias en la ontología que tengan como superclases las encontradas en el punto anterior y cumplan el resto de propiedades del ejemplar original. Esta comprobación se realizará por grupos de propiedades, mediante la división que hemos hecho anteriormente, de forma que al menos se cumpla una de cada grupo diferente. De todos los ejemplares que cumplan las restricciones anteriores, se almacenarán como conceptos extraídos aquellos que también se encuentren en la base de casos.

- En el caso de que no se encuentre ningún ejemplar que cumpla las restricciones de propiedades y de superclases que se imponen, se comprobará si las clases padre, obtenidas en el punto anterior, forman parte o no de la base de casos. En caso de así fuera, se almacenarán como nuevos conceptos encontrados. A partir de aquí nombraremos este comportamiento como una "generalización" de conceptos. Por ejemplo, el concepto "Islas Baleares" es una generalización de los ejemplares "Mallorca" y "Menorca".

- Si aún así no se han obtenido conceptos similares al original, se obtendrán las clases padre de las clases padre actuales, es decir, las clases abuelo de la instancia con la que estamos trabajando o, lo que es lo mismo, las clases que se encuentren dos niveles por encima del actual (siempre en una relación directa). En este punto también es necesario comprobar si estas nuevas clases con las que vamos a trabajar tienen definidas restricciones sobre alguna propiedad . Si así es, las propiedades sobre las cuales están éstas definidas serán eliminadas de la lista de propiedades a cumplir en la búsqueda de nuevos ejemplares. Ésto se debe a que, al generalizar (ascender por la jerarquía), las restricciones pueden dejar de tener efecto, por lo que las propiedades sobre las que éstas se refieren no han de ser tenidas en cuenta más.

  Por ejemplo, la propiedad "hasRating" con el valor "FourStars" (4 estrellas) tiene sentido bajo la clase padre "DeluxeAccommodation" (que implica que todos sus alojamientos tienen 4 estrellas), pero deja de tenerlo si ascendemos a su clase abuelo, "FirstClassAccommodation", ya que para esta clase sus ejemplares pueden tener 4 ó 5 estrellas, y no sólo 4 como en el caso anterior. Por tanto, esta propiedad tendría que ser eliminada a la hora de buscar instancias relacionadas.

  Con estos nuevos padres se volverá a comprobar si alguna de sus instancias hijo cumplen las propiedades requeridas. En caso de que así sea serán almacenadas.

- A partir de este punto, y mientras no se encuentren ejemplares equivalentes, se seguirán obteniendo las clases padre (siempre teniendo en cuenta posibles restricciones que impliquen eliminar determinadas propiedades) de las clases padre actuales, es decir, se seguirá ascendiendo por la jerarquía, hasta que se encuentren conceptos similares. Esta acción se realizará hasta llegar a una clase raíz, sin ser incluida ésta, debido a que al ser la clase más general y pedir todas sus instancias se obtendrían todos los ejemplares posibles.

Todos los pasos anteriores se han de realizar para cada uno de los conceptos, que sean instancias, encontrados por las etapas previas del sistema CBR en la consulta textual del usuario. Una vez que se ha realizado todo este procesamiento, los nuevos conceptos similares encontrados serán almacenados de

nuevo en los atributos FEATURES de cada párrafo correspondiente, es decir, en los párrafos donde se encontraba el concepto original que se utilizó para obtener esta nueva información.

De esta forma, si no existe una coincidencia exacta entre los conceptos encontrados por el sistema en la consulta textual del usuario y los que forman parte de la base de casos, se buscará en la ontología información relacionada y similar. Si ésta es completa y está bien diseñada, está asegurado un resultado óptimo con la aplicación del algoritmo anterior.
Es importante destacar que en la búsqueda de instancias del algoritmo anterior sólo se considera cada ejemplar una sola vez por cada similitud buscada, de forma que así se consigue una mayor efectividad a la hora de realizar búsquedas de nuevos conceptos en la ontología.

Para demostrar el buen comportamiento del algoritmo anterior, consideremos el siguiente ejemplo. El usuario escribe en su consulta el texto "I want to travel to Fuerteventura". Las etapas previas de análisis obtendrán el concepto "Fuerteventura" para el atributo "Region" de la estructura de casos. Al llegar a la etapa "Ontology Layer", el sistema extraerá este concepto y comprobará si existe o no como valor en la base de casos. Como no es así, verificará que éste sí existe en la ontología y que se trata de un ejemplar con las siguientes propiedades:

- Subclase de (o "de tipo") las clases "CanaryIslands", "Spain", "Europe", "Destination" y "Thing" (superclase de todas las clases).
- Propiedad "hasActivity" con valor "Bathing".
- Propiedad "hasActivity" con valor "Sunbathing".
- Propiedad "hasAccommodation" con valor "SolGorriones".
- Propiedad "hasAccommodation" con valor "IberostarPlayaGaviotas".

El siguiente paso será eliminar las propiedades exclusivas. En este caso, son exclusivas las dos propiedades "hasAccommodation", ya que estas instancias sólo las tiene este concepto (estos hoteles sólo están situados en Fuerteventura). Además, se obtendrían las clases padre directas entre todas las superclases obtenidas en el punto anterior, quedando así la lista de propiedades:

- Subclase directa de "CanaryIslands".
- Propiedad "hasActivity" con valor "Bathing".
- Propiedad "hasActivity" con valor "Sunbathing".

La próxima tarea a realizar es la búsqueda de nuevos ejemplares que cumplan estas tres propiedades anteriores. Si miramos en la ontología podemos descubrir que existen varias instancias que las cumplen, como son:

- Tenerife
- GranCanaria
- Lanzarote
- LaPalma
- ElHierro
- LaGomera

De estos seis nuevos ejemplares sólo dos, "Tenerife" y "GranCanaria", también se encuentran en la base de casos, por lo que serían éstos exclusivamente los que se almacenarían como nuevos conceptos extraídos, sustituyendo así a "Fuerteventura".

En el caso de que no se hubieran obtenido nuevos ejemplares en el paso anterior, o que de los encontrados ninguno estuviera también en la base de casos (incluidas las clases padre), lo siguiente sería

buscar las clases abuelo de la instancia “Fuerteventura” o, lo que es lo mismo, las clases padre directas de la clase “CanaryIslands”, siendo ésta el concepto “Spain”. El siguiente paso sería entonces buscar nuevos ejemplares que fueran subclase de “Spain” y con las propiedades “hasActivity” anteriores.

Nuevas utilidades añadidas

La principal utilidad añadida en esta cuarta aproximación del proyecto ha sido la implementación de un método de búsqueda información relativa en ontologías, aunque también se ha desarrollado una ontología propia que podrá ser distribuida en una versión del entorno jCOLIBRI.

Las principales utilidades añadidas en esta etapa son las siguientes:

- Implementación de un método de interacción con ontologías.
- Desarrollo de una ontología propia para el dominio específico de los viajes.
- Tratamiento general de ontologías por el método de interacción.

Implementación de un método de interacción con ontologías

Como ya hemos explicado antes, la principal utilidad añadida al proyecto en esta cuarta aproximación es la implementación de un método de interacción con ontologías, que permite extraer información relativa o similar sobre un concepto buscado.

Este método es utilizado cuando un cierto concepto extraído del texto de consulta del usuario no es encontrado en la base de casos del sistema. De esta forma, puede buscar información similar que permita generar una respuesta que probablemente satisfaga las preferencias del usuario, a modo de recomendación.

Desarrollo de una ontología propia para el dominio específico de los viajes

Además de la implementación del método anterior, en esta aproximación se ha desarrollado una ontología con conocimiento específico del dominio de viajes, en la que se almacena una gran cantidad de información de forma estructurada.

Para el desarrollo de esta ontología se invirtió una gran cantidad de tiempo con el objetivo que tuviera una estructura sólida bien formada sobre la que probar el método de interacción anterior, pero a la vez que fuera sencilla y fácilmente entendible.

Otra de las ventajas de esta ontología es que sobre ella se puede seguir construyendo y añadiendo nueva información que permita completar el conocimiento en el dominio de los viajes.

Tratamiento general de ontologías por el método de interacción

El método de interacción creado en esta etapa se ha desarrollado de forma general e independiente de todo dominio, lo que permite trabajar con cualquier ontología, con la única restricción de que tiene que estar escrita en lenguaje OWL.

De esta forma, cualquier ontología provista por el usuario, puede servir para extraer información relativa sobre un concepto buscado, siempre que ésta esté bien construida y contenga datos relevantes.

El método de interacción accede a la ontología y se mueve por ella independientemente del número de subclases o superclases que tenga un cierto concepto o el número de propiedades de que disponga. El algoritmo anterior está implementado para que este tratamiento se realice de forma genérica.

Limitaciones del sistema

Con el trabajo realizado en esta cuarta aproximación de nuestro proyecto se ha conseguido eliminar una de las principales limitaciones que presentaba el sistema desde sus inicios, como era la imposibilidad de obtener información relacionada con un cierto concepto. Con la implementación del método de interacción con ontologías se ha podido superar esta limitación que otorga al sistema una mayor flexibilidad y un mejor comportamiento de cara a sus usuarios. Aún así, surgen algunas otras pequeñas limitaciones que a continuación pasamos a enumerar:

- Coincidencia entre nombres de conceptos para la ontología y la base de casos.

Algunas de las limitaciones que aún permanecen en el sistema son:

- El sistema no permite la representación de negaciones, conjunciones y disyunciones.

Coincidencia entre nombres de conceptos para la ontología y la base de casos

La principal limitación que presenta esta nueva aproximación es que obliga a que exista una coincidencia de nombres para los conceptos de la base de casos y los de la ontología. De esta forma, los conceptos extraídos de la consulta del usuario pueden ser buscados en la ontología y, a partir de éstos, extraer información relevante si es necesario.

En el caso de que estos nombres no coincidan, el método de interacción buscará el concepto en la ontología y no lo encontrará, por lo que su funcionalidad se verá muy reducida.

Por tanto, la única exigencia de este método es la coincidencia de nombres para conceptos que se refieren a lo mismo. Por ejemplo, el concepto "Coche" puede tener una gran variedad de nombres diferentes, como pueden ser "Vehículo", "Automóvil"... Aunque todos se refieren al mismo concepto, el método de interacción necesita que los nombres utilizados en la base de casos y en la ontología coincidan.

Problemas encontrados

En esta aproximación hemos añadido una nueva utilidad muy importante a nuestro sistema CBR, con la implementación de un método de interacción con ontologías que permite la búsqueda y extracción de información relativa o similar a un concepto buscado. Pero el trabajo realizado no ha estado libre de problemas, ya que han surgido ciertas dificultades y dudas en ciertos aspectos.

Los principales problemas a los que nos hemos enfrentado en esta cuarta aproximación han sido:

- Dificultad en el diseño de la ontología.
- Dificultad en el diseño del comportamiento del método de interacción.
- Adaptación al entorno Java de tratamiento de ontologías.

Dificultad en el diseño de la ontología

Este fue uno de los principales problemas al comienzo de esta cuarta aproximación del proyecto. Era necesario que la ontología que diseñáramos estuviera bien estructurada, fuera completa y resultara sencilla de entender. Además, era necesario que a partir de ella se pudiera extraer información relativa o similar dado un determinado concepto, es decir, que también fuera compatible con el método de interacción que íbamos a diseñar a continuación.

Para realizar este trabajo se buscaron y consultaron un gran número de ontologías que nos pudieran servir de ejemplo e incluso se consideró la opción de reutilizar alguna de ellas, opción que más tarde se descartó porque ninguna cubría completamente nuestras necesidades.

Otra dificultad fue comenzar a conocer el entorno de Protégé, que nos permitiría desarrollar nuestra ontología. Aunque su interfaz es bastante sencilla e intuitiva, siempre se plantean ciertas dudas a la hora de realizar determinadas acciones, como por ejemplo la definición de restricciones en algunas clases o la inferencia y clasificación de ejemplares bajo ciertas clases, tarea que necesita la intervención de un razonador.

Dificultad en el diseño del comportamiento del método de interacción

Sin duda ésta fue la mayor de las dificultades encontradas en esta aproximación. El método que queríamos implementar debía ser totalmente general e independiente de todo dominio para que pudiera interactuar con cualquier ontología OWL posible.

Nuestras dudas principales surgieron a la hora de definir el comportamiento de este método, es decir, las acciones que debía realizar dependiendo del concepto sobre el que se buscaba información, aunque poco a poco se fueron perfilando y decidiendo.

Adaptación al entorno Java de tratamiento de ontologías

Java proporciona una serie de clases que nos permiten la interacción con ontologías: representación de clases y ejemplares, representación de propiedades, operaciones para realizar búsquedas de conceptos, para listar subclases y superclases...

Para poder utilizar toda esta infraestructura fue necesario un largo periodo de tiempo de entrenamiento que nos permitiera adaptarnos a ella y conocer todos los detalles de su funcionamiento.

En muchas ocasiones, al no conocer de forma detallada este entorno, se nos planteaban dudas sobre si las acciones que queríamos realizar era posible llevarlas o no a cabo. Este desconocimiento nos llevó muchas veces a buscar otras alternativas de implantación que si pudiéramos realizar con nuestros conocimientos.

Éstos fueron algunos de los detalles que nos plantearon mayores dificultades en este entorno:

- Obtención de las superclases (o clases padre) de un determinado ejemplar.
- Forma de obtener las clases con restricciones definidas en sus propiedades.
- Tratamiento de las propiedades definidas para una determinada instancia.

Pruebas realizadas

En este apartado se muestran algunas de las pruebas que se han realizado en esta fase de desarrollo del proyecto para comprobar el correcto funcionamiento de las nuevas funcionalidades añadidas. En este caso, nos centraremos en la interacción del sistema con la ontología de viajes y se explicará cómo se extrae la información relacionada que es obtenida como respuesta. En cada una de las consultas que a continuación se listan se ha subrayado la información relevante que debe ser extraída, para facilitar la comprensión del funcionamiento del sistema.

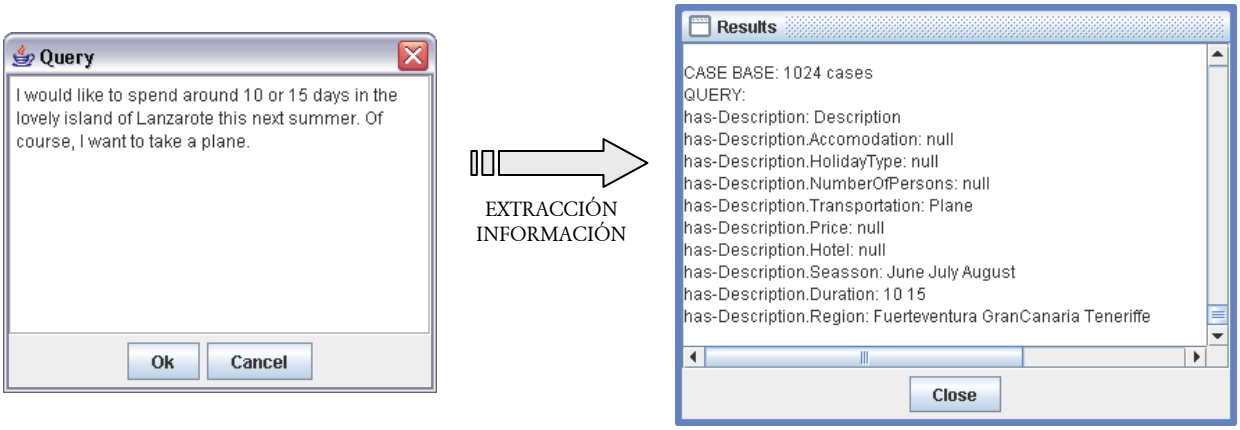
Prueba 1

Consulta

- I would like to spend around 10 or 15 days in the lovely island of Lanzarote this next summer. Of course, I want to take a plane.

Información extraída

- Transportation = Plane
- Season = June July August
- Duration = 10 15
- Region = Fuerteventura GranCanaria Teneriffe



En este primer caso, el sistema es capaz de extraer todos los datos relevantes de la consulta textual del usuario. En la capa de ontologías ("Ontology Layer") el sistema detecta que el destino "Lanzarote" no se encuentra en la base de conocimiento del CBR, por lo que comienza una búsqueda de información similar o relacionada. Para llevar a cabo esta tarea, el sistema localiza el concepto "Lanzarote" dentro de la ontología y busca otros ejemplares que cumplan las misma propiedades que éste tiene, excepto para las propiedades exclusivas, como "hasAccommodation".

El sistema, entonces, determina que otros ejemplares relacionados son "Fuerteventura", "GranCanaria", "Tenerife", "LaPalma", "ElHierro" y "LaGomera" (ver ontología completa en el Apéndice A). Antes de extraer estos datos como información relacionada, se comprueba que existen en la base de casos, es decir, que realmente existen viajes disponibles con estos destinos. Tras realizar dicha comprobación, se almacenan los conceptos "Fuerteventura", "GranCanaria" y "Teneriffe".

Además de la información extraída referente al atributo "Region" (destino del viaje), el sistema también utiliza la etapa de ontologías para obtener los valores "June", "July" y "August" en el campo "Season".

En la segunda fase de desarrollo del proyecto, estos valores eran definiditos como sinónimos del usuario del concepto "Summer", y al ser localizado éste en una consulta era sustituido por los tres primeros. De esta forma, se podía determinar de manera sencilla que la estación de verano se corresponde con los meses de Junio, Julio y Agosto. En esta cuarta fase se han eliminado esta definición de sinónimos y se utiliza el conocimiento contenido en la ontología para obtener la misma información. De esta forma, una vez reconocido el concepto "Summer" en la consulta del usuario, es localizado dentro de la ontología por la nueva etapa del sistema, descubriendo de esta forma que se trata de una clase y no de un ejemplar. Así, la acción a realizar es extraer todos los individuales que forman parte de esta clase, siendo éstos los conceptos "June", "July" y "August".

Este comportamiento del sistema se conoce con el nombre de "especialización de conceptos". Como el concepto "Summer" no se encuentra en la base de conocimiento, es sustituido por otros valores más especializados, como son "June", "July" y "August". Otra situación similar podría darse si el sistema extrajese de la consulta del usuario el concepto "Francia" como valor del atributo "Destination" y en la base de conocimiento del CBR sólo se considerasen como destinos de viajes ciudades en vez de países. De esta forma, el sistema especializaría el concepto "Francia" y obtendría todas las ciudades de este país que estuviesen presentes en la base de casos.

El resto de datos se extraen del mismo modo que se ha explicado en las pruebas realizadas en las fases anteriores del sistema.

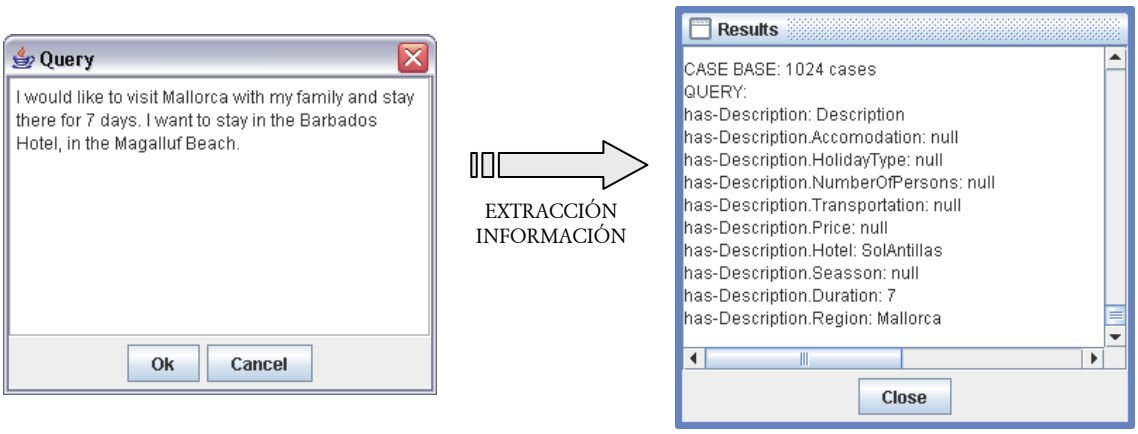
Prueba 2

Consulta

- I would like to visit Mallorca with my family and stay there for 7 days. I want to stay in the Barbados Hotel, in the Magalluf Beach.

Información extraída

- Hotel = SolAntillas
- Duration = 7
- Region = Mallorca



En esta segunda prueba, la nueva etapa de ontologías actúa sobre la información encontrada para el atributo "Hotel". El usuario ha especificado que desea alojarse en el hotel "Sol Barbados" en Mallorca. Como se puede comprobar, este valor no se encuentra en la base de conocimiento del CBR, por lo que el sistema utiliza la capa de ontologías para encontrar información relacionada. De la misma forma que

en la prueba anterior, primero se localiza el concepto y después se extraen todas las propiedades que lo componen, para pasar a buscar otros ejemplares que también las cumplan. Así, el sistema localiza el hotel "Sol Antillas", localizado también en Mallorca, en una zona de playa (ambos son hoteles de costa) y con el mismo número de estrellas que el anterior (cuatro). Tras comprobar que este valor sí se encuentra en la base de casos, el sistema lo almacena como dato extraído para el atributo "Hotel".

En este caso, se puede comprobar incluso en un folleto de vacaciones que ambos hoteles se encuentran el uno junto al otro, en la playa de Magalluf.

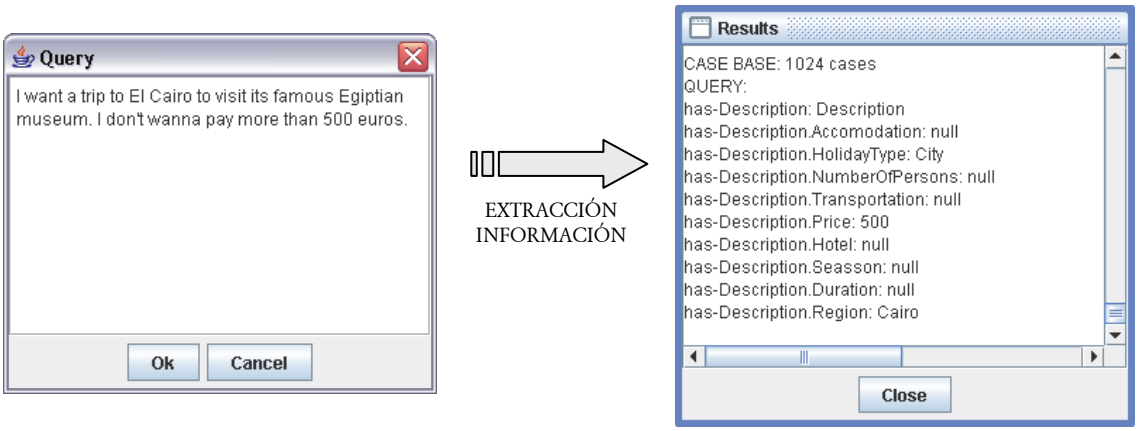
Prueba 3

Consulta

- I want a trip to El Cairo to visit its famous Egiptian museum. I don't wanna pay more than 500 euros.

Información extraída

- HolidayType = City
- Price = 500
- Region = Cairo



En este último ejemplo, el sistema utiliza la etapa de ontologías sobre el valor extraído para el atributo "HolidayType". El usuario ha especificado en su consulta que le gustaría ver el famoso "Museo Egipcio" de la ciudad de El Cairo. En un primer momento, el sistema extrae en la capa "Feature Value Layer" el concepto "Museums" (museos) para el atributo "HolidayType" (tipo de vacaciones). Cómo este valor no se encuentra en la base de conocimiento del CBR, la capa de ontologías debe buscar información relacionada que permita generar una respuesta que satisfaga las necesidades del usuario. De esta forma, el sistema localiza el concepto "Museums" en nuestra ontología de viajes y descubre que se trata de un ejemplar de la clase "City", del que extrae sus propiedades y realiza una búsqueda para encontrar instancias semejantes bajo dicha clase. Tras no encontrar resultados en esta primera búsqueda, el sistema comprueba si el propio concepto "City" se encuentra en la base de casos del CBR. Como es así, éste es almacenado como dato extraído para el campo "HolidayType" de la consulta estructurada.

Este procedimiento se denomina "generalización de conceptos", ya que el sistema, al no encontrar como tipo de vacaciones el visitar un museo ("Museums"), comprueba si su concepto padre "vacaciones de ciudad ("City") es considerado así. Como este valor si se encuentra en la base de conocimiento del CBR, este segundo concepto es almacenado en sustitución del primero.

Otra situación similar a la anterior sería, por ejemplo, si el sistema extrajera de la consulta "París" como valor para el atributo "Destination" y la base de conocimiento sólo considerara como destinos los países y no las ciudades. De esta forma, el sistema generalizaría el concepto "París" y obtendría "Francia" como destino del viaje del usuario.

Apartado 5
Estudio de resultados

Estudio de resultados

En este apartado se realiza un estudio del comportamiento de nuestro sistema a través de los resultados obtenidos para diferentes consultas. Para llevar a cabo este test experimental se han recopilado una gran cantidad de consultas textuales sobre el dominio de viajes, para ser ejecutadas en nuestro sistema y, así, poder analizar los resultados generados por éste.

Este estudio trata de medir la efectividad media de nuestro sistema para cada consulta que recibe. Esta medida se basa en la idea de medir la proporción que existe entre el número de datos relevantes que contiene una determinada consulta y los que nuestro sistema detecta y extrae correctamente.

La especificación formal de dicha proporción es la siguiente:

$$\text{Efectividad} = \frac{\text{Número de datos extraídos correctamente por el sistema}}{\text{Número total de datos relevantes en la consulta}}$$

Consideremos el siguiente caso a modo de ejemplo. Supongamos que un usuario especifica en su consulta que desea viajar a Egipto en avión. En este caso, el número de datos relevantes es dos, correspondientes al destino y al transporte. Si nuestro sistema es capaz de reconocer y extraer ambos valores, su efectividad para esta consulta será del 100% (2/2). Si sólo lo hace con uno de los dos será del 50% (1/2) y del 0% (0/2) si no consigue extraer ningún dato.

Para el caso de la cuarta fase de desarrollo, correspondiente a la utilización de ontologías, se varía la medida de efectividad a tener en cuenta, debido a que esta capa obtiene información aproximada y no exacta. En este caso, si la información recuperada por el sistema es adecuada, es decir, está relacionada o es similar, se considerará que el dato correspondiente ha sido reconocido correctamente. En caso contrario, la consideración será al revés. Por ejemplo, si para el destino Menorca, que no se encuentra en la base de conocimiento, el sistema recupera como similares “Mallorca” e “Ibiza”, su efectividad para ese dato será del 100%. Si por el contrario recupera “Egipto” (o ninguno), que no tiene que ver, el resultado será del 0%.

Los resultados obtenidos en este estudio marcan un comportamiento general de nuestro sistema, que puede ser muy bueno para determinadas consultas y muy malo para otras. A continuación se describen algunos ejemplos de consultas con diferentes resultados:

- “I would like to visit the wonderful place of Cairo the next month of July. I want to stay there 10 days. Price around 1000 euros for 2 people.”

  Al procesar esta primera consulta, el sistema obtiene un **100%** de efectividad, ya que es capaz de extraer toda la información relevante que ésta contiene: destino El Cairo, época Julio, duración 10 días, precio 1000 euros y número de personas 2.

- “I want a trip to the Fuerteventura for 15 days. I want to go with my girlfriend. The price has to be less than 1000 euros.”

  En esta segunda consulta, el sistema presenta un término medio de comportamiento, ya que obtiene un **50%** de efectividad. En este caso es capaz de obtener que el destino del viaje es la Fuerteventura y que la duración es de 15 días. Por el contrario, no reconoce que el número de personas es dos (la persona que escribe y su novia) y que el precio debe ser menor de 1000 euros. Para el primero de estos dos valores, el sistema no sería capaz de obtener información

alguna, pero para el segundo lo interpretaría de forma errónea y llegaría a la conclusión de que el precio del viaje es de 1000 euros, cuando en realidad ha de ser menor.

- "I would like to book a trip to spend Friday and Saturday nights having a good time. I don't mind the destination. I don't want to take a plane."

  Para esta última consulta, el sistema no puede extraer ningún dato relevante, es decir, obtiene un **0%** de efectividad. Debería ser capaz de deducir que el usuario ha especificado que desea un viaje de dos días (noches del viernes y del sábado), que le da igual el destino (es decir, que puede ser cualquiera de los todos los posibles) y que no quiere ir en avión. Para los dos primero datos, el sistema no es capaz de obtener ninguna información, pero para el tercero almacenaría como medio de transporte el avión, ya que no es capaz de procesar las negaciones, y lo toma como una afirmación.

Con este estudio es posible verificar que nuestro sistema presenta un comportamiento medio óptimo para la mayoría de las consultas típicas que puede recibir. Pero se han encontrado ciertos patrones que contienen datos relevantes pero que no pueden ser reconocidos por nuestro sistema, y que, aunque no son muy frecuentes en las consultas de los usuarios, constituyen sus mayores puntos débiles. A continuación se destacan los principales, traducidos al español para una mejor comprensión:

- "La vuelta del viaje a finales del mes que viene". En este caso, el sistema debería ser capaz de deducir con esta frase la duración del viaje, teniendo en cuenta la fecha actual y que la vuelta ha de ser a finales del siguiente mes.

- "Deseo ir con mi mujer y mis cuatro hijos". Aquí, el sistema debería de obtener como valor del atributo "número de personas" el valor 6, pero es incapaz.

- "No quiero ir a Mallorca". Como ya hemos comentado antes, las negaciones no pueden ser reconocidas por nuestro sistema, que, en este caso, extraería Mallorca como destino del viaje, cuando el sentido de la consulta es totalmente el contrario.

- "Pasar allí las noches del viernes y el sábado". Este es uno de los ejemplos de las consultas anteriores, en la que el sistema debería de obtener como duración del viaje el valor 2.

- "El precio debe ser menos de 500 euros". En este caso, el sistema extraería como precio del viaje el valor 500, cuando el usuario está indicando que ha de ser menor.

- "Dos grupos de 10 personas". Para este caso, el sistema debería de extraer 20 como valor para el atributo "número de personas", pero, en realidad, se comporta erróneamente y almacena 10 (número que precede a la palabra "personas").

- "Tres semanas de viaje". El sistema debería obtener en este caso el valor 21 como duración del viaje, pero no es capaz de extraer ninguna información, ya que espera la representación en días y no en semanas.

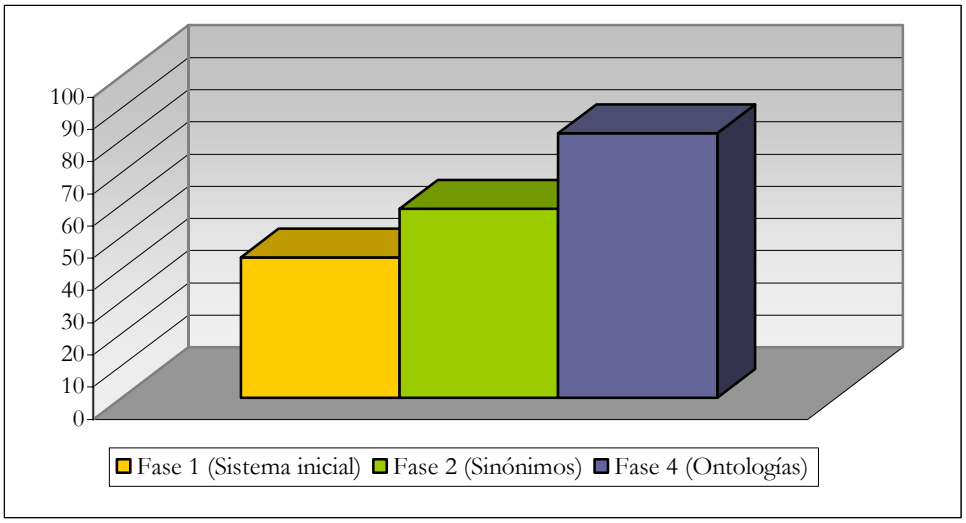
- "Quiero ir a un pueblo cerca de Madrid". En este último caso, el sistema no es capaz de extraer la información precisa que se está pidiendo, ya que sólo permite extraer como dato relevante la ciudad de Madrid, cuando el usuario está especificando otro lugar.

La finalidad de este estudio es comprobar la mejora que se produce en los resultados del sistema en cada una de las diferentes fases de desarrollo en las que se divide: sistema inicial, sinónimos y ontologías. La fase de confirmación de datos no se va a tener en cuenta debido a que no introduce mejoras en la capa de extracción de información, sino que sólo permite al usuario verificar los valores obtenidos.

Es importante destacar que para la realización de este estudio de resultados no se ha tenido en cuenta la posible falta de conocimiento del sistema, ya que es el propio desarrollador el que debe incluir cuánto más mejor para hacerlo lo más completo posible. De esta forma, los resultados obtenidos en este estudio representan el comportamiento de nuestro sistema suponiendo que dispone del conocimiento requerido a la hora de analizar cada una de las consultas aplicadas, es decir, indican si sería capaz de extraer los datos relevantes si dispusiera del conocimiento necesario.

Para llevar a cabo una primera medición de los resultados obtenidos se han probado en el sistema 20 consultas de diferentes tipos para cada una de las fases de desarrollo, excepto para la tres. Estas consultas contienen exclusivamente información representada por nuestro sistema, es decir, con datos referentes a los atributos que nuestro sistema procesa (los que forman la estructura del caso): destino, transporte, número de personas, precio, época, etc. El objetivo de este primer estudio es ver qué datos extraería nuestro sistema para cada uno de esos atributos, comparándolo este comportamiento con el que tendría una persona cualquier con la misma consulta, es decir, si ambos, sistema y usuario, reconocerían los mismos datos para todos los atributos.

El siguiente gráfico muestra los resultados obtenidos tras la ejecución de todas las consultas anteriores en cada una de las fases de desarrollo del sistema. Cada una de las barras muestras la efectividad media del sistema en cada fase correspondiente. Al tomar la media aritmética como dato relevante estamos considerando el comportamiento general del sistema, que, como ya hemos dicho antes, puede ser muy bueno para ciertas consultas y muy malo para otras.

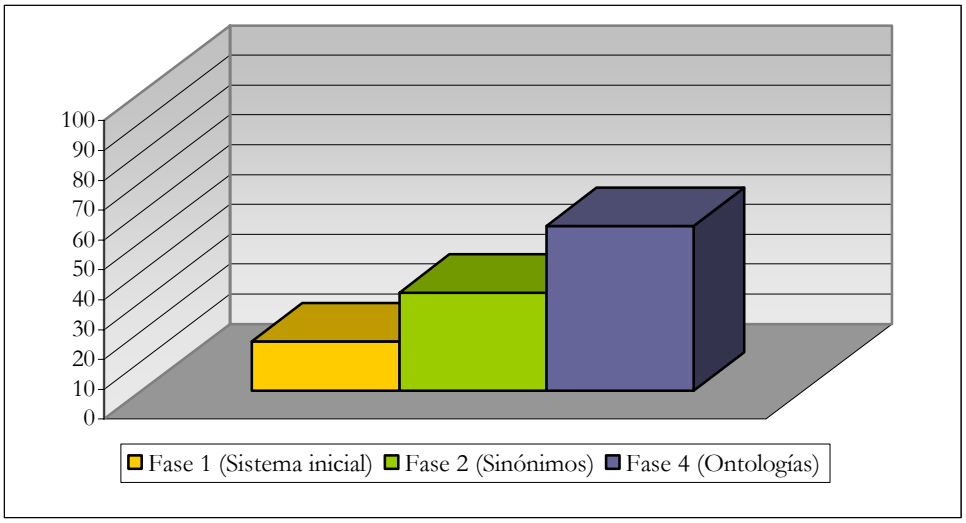


Observando el gráfico anterior es posible darse cuenta que los resultados de nuestro sistema mejoran para cada una de las diferentes fases de desarrollo, obteniendo mejores datos de efectividad para la etapa de ontologías y los peores para la versión inicial. Ésto es evidente, ya que cada nueva fase incorpora las funcionalidades de la anterior, pero demuestra que las utilidades desarrolladas mejoran el rendimiento de nuestro sistema de forma significativa.

También es importante destacar el dato de efectividad de la fase 4, que ronda el 82%. Este valor indica que nuestro sistema es capaz de extraer la mayoría de los datos relevantes que contiene una consulta, en promedio. Es importante volver a destacar en este punto que este comportamiento depende de la consulta que se analice, ya que para unas se obtendrán mejores resultados que para otras. Aún así, los resultados obtenidos son muy buenos en el ámbito en el que estamos trabajando. Como hemos dicho antes, los patrones que nuestro sistema no es capaz de reconocer no son muy frecuentes, lo que permite que los resultados obtenidos sean bastante buenos.

Para llevar a cabo una segunda medición del comportamiento de nuestro sistema, se han ejecutado en él una segundo conjunto de otras 20 consultas de diferentes tipos, esta vez realizadas por terceras personas ajenas a este proyecto que desconocían totalmente su comportamiento, para evitar así que nuestras pruebas fueran subjetivas en este segundo caso. Al igual que para el estudio anterior, la ejecución de estas consultas se ha realizado en las tres diferentes fases de desarrollo del sistema, de forma que se observe la mejoría en los resultados que cada una de ellas proporciona.

En el siguiente gráfico se muestra la efectividad media del sistema obtenida para las consultas anteriores en cada una de las fases de desarrollo:



Si examinamos el gráfico anterior podemos comprobar que en este caso los resultados obtenidos son peores que en el primer estudio, ya que las consultas son generales y no se centran en el ámbito en el que se ha diseñado nuestro sistema. Aún así, se obtiene un valor cercano al 55%, que puede considerarse como un dato bueno si tenemos en cuenta las grandes limitaciones que presenta el tratamiento del lenguaje natural.

Igual que en el estudio anterior, los resultados obtenidos para nuestro sistema mejoran para cada una de las fases de desarrollo en las que se ha dividido. De esta forma, el peor dato se obtiene para primera fase, correspondiente al diseño inicial del sistema y en la que no se incluyen utilidades adicionales, y el mejor para la cuarta, correspondiente a la etapa de uso de ontologías. Es necesario destacar también en este punto que el comportamiento de nuestro sistema podría mejorar con la inclusión de más atributos en la estructura del caso (como, por ejemplo, el régimen del alojamiento: todo incluido, pensión completa...), ya que muchas de estas consultas se refieren a datos que no son representables por nuestro sistema.

Por último, podemos comentar que se podrían obtener mejores resultados de efectividad para el sistema al añadirle nuevas funcionalidades, que permitan extraer y representar más información de las consultas de los usuarios. La finalidad sería conseguir un dato mayor que el obtenido para la fase 4.

Apartado 6
Conclusiones

Conclusiones

A lo largo de este documento se ha ido explicando el desarrollo que ha tenido nuestro proyecto durante todas las fases que consta. En este último apartado se va a realizar una valoración general del trabajo llevado a cabo y se van a destacar los principales objetivos logrados.

Como ya hemos visto, el principal trabajo realizado en este proyecto ha sido el tratamiento del lenguaje natural en la etapa de extracción de información sobre las consultas de los usuarios. Según se describió, este proceso presenta una gran cantidad de problemas y dificultades que restan precisión e impiden en ocasiones que nuestro sistema funcione de forma correcta, como, por ejemplo, la ambigüedad o la no completitud de las oraciones. Ésto hace que el proceso de desarrollo de nuestro sistema no fuera algo fácil, debido a que el problema a resolver no era trivial. Fue preciso, por tanto, tener en cuenta en todo momento las limitaciones que se nos presentaban y el ámbito reducido en el que íbamos a trabajar, para poder intentar conseguir dentro de él los mejores resultados posibles.

Con el diseño de la arquitectura de nuestro sistema se intentó crear una estructura que fuera reusable y adaptable a todo tipo de dominios específicos. De esta forma, la etapa de "Búsqueda de Respuesta" puede ser llevada a cabo por cualquier módulo que genere, a partir de una representación estructurada de datos, una respuesta adecuada. En nuestro caso, se eligió en entono jCOLIBRI debido a las enormes ventajas que presentaba, como, por ejemplo, su sencillez o su capacidad para poder crear sistemas CBR para cualquier tipo de dominio, a las que hay que añadir las que nos proporciona el razonamiento basado en casos frente a otro tipo de disciplinas (consultas contra bases de datos...).

Además, el entorno jCOLIBRI ya proporciona una extensión textual compuesta por una serie de métodos que nos permiten llevar a cabo diferentes técnicas de procesamiento de textos: filtro de palabras, glosario, identificación de expresiones propias, clasificación por temas, etc. Gracias a esta utilidad, jCOLIBRI se convirtió en el entorno perfecto para, además de implementar nuestra etapa de generación de respuesta, desarrollar todo nuestro sistema por completo mediante su división en diferentes tareas, como ya se ha visto en cada una de las fases que hemos llevado a cabo.

En cada una de estas fases de desarrollo hemos ido añadiendo nuevas funcionalidades a nuestro sistema, pero también al entorno jCOLIBRI, en forma de métodos y tareas reutilizables en otros. En la primera fase se desarrolló nuestra versión inicial del sistema que fue refinándose en las posteriores etapas, con la utilización de sinónimos (de WordNet y definidos por el desarrollador) y ontologías, y la interfaz gráfica de confirmación de datos relevantes extraídos. Estas nuevas funcionalidades desarrolladas a lo largo de todo el proyecto son genéricas e independientes de todo dominio, por lo que pueden ser utilizados para cualquier otro tipo de sistema. Aunque nosotros tomamos como dominio de trabajo el de los viajes, ésto no ha impedido que la implementación de nuestras tareas se haya hecho de forma independiente a él. Ésta es una de las principales ventajas de las utilidades que hemos desarrollado en nuestro proyecto, que permite que puedan ser utilizadas para cualquier otro sistema sobre un tipo diferente de dominio.

La principal utilidad añadida al entorno jCOLIBRI consiste en una interfaz textual para sistemas CBR. Se trata de una tarea encargada de mostrar al usuario, durante la etapa de obtención de la query, una pequeña ventana en la que puede escribir la descripción textual sobre el dominio con el que se trabaje. Una vez enviada esta consulta, el sistema la procesa y extrae a partir de ella una serie de datos relevantes que son almacenados en una query, que servirá como entrada para el CBR para poder encontrar una solución adecuada. Esta infraestructura añadida a jCOLIBRI permite realizar consultas textuales sobre sistemas CBR, funcionalidad que no estaba disponible hasta el momento, ya que sólo se permitía seleccionar los datos directamente desde un formulario. De esta forma, ahora también se dispone de una interfaz textual para el entorno.

A partir de este diseño inicial del sistema se fueron desarrollando nuevas utilidades, todas ellas con la capacidad de ser reutilizables. En primer lugar se creo una etapa de sinónimos que utiliza la base léxica

de WordNet y que tiene como objetivo buscar coincidencias entre las palabras que componen las consultas de los usuarios y los valores que se almacenan en la base de conocimiento del sistema CBR. Esta nueva etapa también incluye una capa de reconocimiento de sinónimos definidos por el propio desarrollador, y que permite encontrar en los textos de consulta expresiones específicas del dominio durante la fase de extracción de información. Estos sinónimos son definidos por los usuarios del entorno en unos ficheros de configuración especiales que incluyen una serie de reglas o patrones para el reconocimiento de dichas expresiones. De esta forma, si cambiamos el dominio de trabajo, sólo será necesario cambiar las reglas que componen estos ficheros para poder adaptar el nuevo sistema. Por tanto, y una vez más, se ha dotado a este componente de una completa generalidad con el fin de poder reutilizarlo para otros fines.

En la tercera fase de desarrollo de nuestro proyecto se llevó a cabo la implementación de una etapa de confirmación de información extraída, consistente en una interfaz gráfica en la que los usuarios del sistema pueden visualizar los datos relevantes que han sido obtenidos a partir del análisis de sus consultas. De esta forma, es posible modificar esta información que será utilizada por el módulo de búsqueda de respuesta para encontrar una solución. El diseño de esta nueva capa también se ha realizado de forma genérica, lo que permite mostrar al usuario todos los datos y valores existentes, con independencia de su tipo o de su número. Así, estamos dotando a nuestro sistema de una flexibilidad total.

Por último, en la cuarta fase de desarrollo del proyecto se llevó a cabo la que puede considerarse, sin duda, la mayor de las utilidades implementadas por nosotros para el entorno jCOLIBRI, consistente en el método de interacción de con ontologías. La finalidad de este método es la búsqueda información similar o relacionada con un cierto concepto dado, de forma que el sistema pueda generar respuestas aproximadas a partir de los datos relevantes obtenidos de la capa de extracción de información. Esta nueva etapa requiere el uso, como es evidente, de una ontología con conocimiento acerca del dominio específico con el que se trabaja. Como el método diseñado es totalmente genérico, su interacción con la ontología se realiza de igual forma para todas las estructuras que pueda presentar, el número de clases o su distribución. Una vez más, la nueva funcionalidad diseñada puede ser reutilizada para crear otro sistema en otro dominio diferente.

Como hemos detallado en los párrafos anteriores, todas las utilidades desarrolladas en nuestro proyecto se han implementado de forma general e independiente de todo dominio, pero perfectamente adaptables a todos. De esta forma, para crear un sistema nuevo con estos componentes, sólo es necesario incluir el conocimiento específico del dominio. Ésta es una de las principales ventajas de nuestro diseño. Además, todas las utilidades desarrolladas pueden ser integradas en jCOLIBRI como parte del entorno, de forma que el resto de la comunidad de usuarios puede también utilizarlas. En este aspecto radica la importancia de su reusabilidad.

En el apartado anterior se han presentado los resultados del estudio realizado en nuestro sistema, sobre el que se han ejecutado una gran cantidad de consultas para comprobar su funcionamiento. En él es posible observar que el comportamiento del sistema mejora con cada utilidad desarrollada. En la primera fase los resultados no son del todo positivos, ya que la capa de extracción de información aún no incorpora funcionalidades más complejas que permitan reconocer características y expresiones propias. Con el desarrollo de la capa de sinónimos, los resultados obtenidos en las pruebas mejorar significativamente, sobre todo gracias a la utilidad que permite definir sinónimos propios. Para la cuarta fase de desarrollo de nuestro proyecto (la tercera no añade nuevas utilidades a la capa de extracción de información) la mejora en los resultados también se hace patente, gracias a que el sistema puede encontrar datos similares o relacionados cuando un determinado concepto extraído de una consulta no se encuentra en la base de conocimiento del CBR. De esta forma, en vez de ignorar este dato, el sistema es capaz de encontrar una alternativa equivalente, lo que permite mejorar su funcionamiento y, por tanto, los resultados obtenidos. Con este estudio es posible ver que los resultados del sistema han ido mejorando a medida que hemos ido incluyendo nuevas funcionalidades, por tanto, su gran utilidad queda demostrada.

A modo de resumen podemos decir que, a pesar de las limitaciones previas, hemos conseguido que el comportamiento de nuestro sistema sea aceptable, habiendo mejorado los resultados con cada nueva utilidad añadida. Es importante también destacar las enormes ventajas que proporcionan los componentes diseñados gracias a su generalidad independiente de todo dominio y totalmente reusables.

Apartado 7
Bibliografía

Bibliografía

[01] F. Escolano, M. A. Cazorla, M. I. Alfonso, O. Colomina and M. A. Lozano. *Inteligencia Artificial. Modelos, técnicas y áreas de aplicación.* Thomson, 2003. ISBN: 84-9732-183-9.

[02] M. De Buenaza, J. M. Gómez and B. Díaz-Agudo. *Using WordNet to Complement Training Information in Text Categorisation.* En: N. Nicolov and R. Mitkov. *Recent Advances in Natural Language Processing II.* John Benjamins B.V., 2000. p. 353-364. ISBN: 90-272-3695-X.

[03] C. Riesbeck and R. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Cambridge, MA, 1989.

[04] J. A. Recio-García, A. Sánchez, B. Díaz-Agudo and P. A. González-Calero. *jCOLIBRI 1.0 in a nutshell. A software tool for designing CBR systems* [en línea]. Group of Artificial Intelligence Applications (GAIA).
<http://gaia.fdi.ucm.es/grupo/publications/2005-ukcbr-jCOLIBRI.pdf>

[05] A. Sánchez, J. A. Recio, B. Díaz-Agudo and P. González-Calero. *Case structures in jCOLIBRI* [en línea]. Group of Artificial Intelligence Applications (GAIA).
<http://gaia.fdi.ucm.es/grupo/publications/2005-AI-sanchez-case.pdf>

[06] J. A. Recio, B. Díaz-Agudo, M. A. Gómez-Martín and N. Wiratunga. *Extending jCOLIBRI for Textual CBR* [en línea]. Group of Artificial Intelligence Applications (GAIA).
<http://gaia.fdi.ucm.es/grupo/publications/2005-iccbr-recio-textual.pdf>

[07] J. A. Recio-García and B. Díaz-Agudo. *An introductory user guide to jCOLIBRI 0.3* [en línea]. Group of Artificial Intelligence Applications (GAIA).
<http://gaia.fdi.ucm.es/grupo/publications/JColibriUserManual.pdf>

[08] N. F. Noy and D. L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology* [en línea]. Stanford University.
<http://protege.stanford.edu/publications/ontology_development/ontology101.pdf>

[09] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe. *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools* [en línea]. The University Of Manchester.
<http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>

[10] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa. *JADE, a white paper* [en línea]. Telecom Italia.
<http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf>

[11] Group of Artificial Intelligence Applications. *GAIA Homepage* [en línea].
<http://gaia.fdi.ucm.es/ >

[12] Group of Artificial Intelligence Applications. *jCOLIBRI: Case Based Reasoning Framework* [en línea].
<http://gaia.fdi.ucm.es/grupo/projects/jcolibri/index.html>

[13] Group of Artificial Intelligence Applications. *SourceForge.net: jCOLIBRI: CBR Framework* [en línea].
<http://sourceforge.net/projects/jcolibri-cbr/>

[14] Belén Díaz-Agudo. *Ingeniería de Sistemas Basados en Conocimiento Curso 2005-2006* [en línea]. Facultad de Informática, Universidad Complutense de Madrid.
<http://www.fdi.ucm.es/profesor/belend/ISBC/isbc.html>

[15] Stanford University. *The Protégé Ontology Editor and Knowledge Acquisition System* [en línea].
<http://protege.stanford.edu/>

[16] Wikimedia Foundation, Inc. *Wikipedia, the free encyclopedia* [en línea].
<http://en.wikipedia.org/wiki/Main_Page>

[17] Tilab. *JADE - Java Agent DEvelopment Framework* [en línea]. Telecom Italia.
<http://jade.tilab.com/>

[18] Kurt J. Hayes and O. Steele. *SourceForge.net: JWordNet* [en línea].
<http://sourceforge.net/projects/jwn/>

[19] S. Martín. *Introducción NLP. Recuperación y Organización de la Información* [en línea].
<http://introduccion-nlp.webcindario.com/>

[20] J. Carbonell. *El procesamiento del lenguaje natural, tecnología en transición* [en línea]. Traducción de G. Arrarte. Instituto Cervantes (España).
<http://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/ponenc_carbonell.htm>

[21] National Institute of Standards and Technology (NIST). *ACE - Automatic Content Extraction* [en línea].
<http://www.nist.gov/speech/tests/ace/>

[22] SRI International's Artificial Intelligence Center. *The SRI International AIC FASTUS System* [en línea].
<http://www.ai.sri.com/~appelt/fastus.html>

[23] Tsuji Laboratory, Department of Information Science, Faculty of Science, University of Tokyo. *GENIA Project* [en línea].
<http://gate.ac.uk/ie/annie.html>

[24] Text Analysis International, Inc. *VisualText* [en línea].
<http://www.textanalysis.com/Products/Overview/overview.html>

[25] F. Ciravegna and A. Lavelli. *The Pinocchio Information Extraction Toolkit* [en línea]. ITC-Irst, Centro per la Ricerca Scientifica e Tecnologica.
<http://tcc.itc.it/research/textec/tools-resources/pinocchio.html>

[26] H. Cunningham, V. Tablen and K. Bontcheva. *GATE, A General Architecture for Text Engineering* [en línea].
<http://gate.ac.uk/ie/ie_example.html>

[27] Natural Language Processing Group, The University of Sheffield. *ANNIE - a robust cross-domain IE system* [en línea].
<http://gate.ac.uk/ie/annie.html>

[28] L. Lozano and J. Fernández. *Razonamiento basado en casos: Una Visión General* [en línea]. Universidad de Valladolid.
<http://www.infor.uva.es/~calonso/IAI/TrabajoAlumnos/Razonamiento basado en casos.pdf>

[29] World Wide Web Consortium. *Web Services Activity* [en línea].
< http://www.w3.org/2002/ws/>

[30] OASIS Open. *OASIS Web Services Business Process Execution Language (WSBPEL) TC* [en línea].
< http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel >

[31] Sun Microsystems, Inc. *JavaServer Pages Technology* [en línea].
<http://java.sun.com/products/jsp/>

Apéndice A
Ontología

Ontología de viajes

En este apéndice se incluye la ontología de viajes completa que hemos creado para nuestro proyecto. Los ejemplares de la ontología se han destacado mediante subrayados, para facilitar su identificación. Las propiedades están marcadas en cursiva, para que sean fácilmente distinguibles. A continuación del siguiente listado se describen las propiedades que relacionan los diferentes ejemplares de la ontología.

- Thing
  - Accommodation
    - AverageAccommodation (=)

      Necessary & Sufficient: ∃ hasRating { HolidayFlat ThreeStars }
      - TrypAlondras (inferido)
    - BudgetAccommodation (=)

      Necessary & Sufficient: ∃ hasRating { TwoStars OneStar }
    - Cottage
    - FirstClassAccommodation (=)

      Necessary & Sufficient: ∃ hasRating { FiveStars FourStars }
      - DeluxeAccommodation (=)

        Necessary & Sufficient: ∃ hasRating { FourStars }
        - IberostarPlayaGaviotas (inferido)
        - MeliaAvenidaDeAmerica (inferido)
        - NHJoseAbascal (inferido)
        - SolAntillas (inferido)
        - SolBarbados (inferido)
        - SolGorriones (inferido)
      - LuxuryAccommodation (=)

        Necessary & Sufficient: ∃ hasRating { FiveStars }
        - GranMeliaVolcanLanzarote (inferido)
        - MeliaDeMar (inferido)
    - Hostel
    - Hotel
      - CityHotel
        - MeliaAvenidaDeAmerica
          - *hasRating* = *FourStars*
          - *isLocatedAt* = *Madrid*
        - MeliaDeMar
          - *hasRating* = *FiveStars*
          - *isLocatedAt* = *Mallorca*
        - NHJoseAbascal
          - *hasRating* = *FourStars*
          - *isLocatedAt* = *Madrid*
        - TrypAlondras
          - *hasRating* = *ThreeStars*
          - *isLocatedAt* = *Madrid*
      - CoastHotel
        - GranMeliaVolcanLanzarote
          - *hasRating* = *FiveStars*
          - *isLocatedAt* = *Lanzarote*
        - IberostarPlayaGaviotas
          - *hasRating* = *FourStars*

        - *isLocatedAt = Fuerteventura*
    - SolAntillas
        - *hasRating = FourStars*
        - *isLocatedAt = Mallorca*
    - SolBarbados
        - *hasRating = FourStars*
        - *isLocatedAt = Mallorca*
    - SolGorriones
        - *hasRating = FourStars*
        - *isLocatedAt = Fuerteventura*
- Parador
    - ParadorAlcalaDeHenares
    - ParadorAstorga
    - ParadorCuenca
    - ParadorSantiago
    - ParadorSosDelReyCatolico
- Resort
    - AllInclusive
    - ClubHotel

- AccommodationRating (=)
  Necessary & Sufficient: {OneStar TwoStars ThreeStars HolidayFlat FourStars FiveStars}
    - OneStar
    - TwoStars
    - ThreeStars
    - HolidayFlat
    - FourStars
    - FiveStars

- Activity
    - Adventure
        - AirActivity
            - Globe
            - HangGliding
            - Parachuting
            - Paragliding
        - GroundActivity
            - MountainActivity
                - SnowCoveredMountainActivity
                    - Skiing
                    - Sledge
                    - Snowboarding
                - Climbing
            - Quads
            - RidingHorses
        - SeaActivity
            - Surf
                - SkySurfing
                - Surfing
                - WindSurfing
            - Diving
            - Snorkeling
    - Cultural
        - City

        - Monuments
        - Museums
    - Education
        - Language
            - LanguageLessons
            - TalkingPractice
    - Gastronomical
        - Drinking
        - Eating
- Recretation
    - Leisure
        - AmusementPark
        - Zoo
    - Show
        - Cinema
        - *Theatre*
- Rest
    - Beach
        - Bathing
        - Sunbathing
    - Spa
        - Masagge
        - Sauna
        - ThermalSpring

- Destination
    - Africa
        - AustralAfrica
            - Namibia
                - Windhoek
            - SouthAfrica
                - CapeCity
        - CentralAfrica
            - Angola
                - Luanda
            - Cameroun
                - Yaoundé
            - RepublicOfTheCongo
                - Brazzaville
        - EasternAfrica
            - Burundi
                - Bujumbura
            - Ethiopia
                - AddisAbeba
            - Kenya
                - Nairobi
            - Madagascar
                - Antananarivo
            - Rwanda
                - Kigali
            - SychellesIslands
                - Victoria
            - Somalia
                - Mogadiscio

        - Tanzania
            - Dodoma
        - Zimbawe
            - Harare
    - NorthAfrica
        - Algeria
            - Algiers
        - Egypt
            - Cairo
        - Libya
            - Tripoli
        - Morocco
            - Rabat
        - Tunisia
            - Tunisia
    - WesternAfrica
        - Ghana
            - Accra
        - IvoryCoast
            - Yamoussoukro
        - Mali
            - Bamako
        - Nigeria
            - Abuja
        - Senegal
            - Dakar
        - Togo
            - Lomé
- America
    - NothAmerica
        - Canada
            - Ottawa
        - USA
            - WashingtonDC
            - NewYorkCity
            - LosAngeles
            - LasVegas
            - Miami
            - SanFranciso
            - SanLuis
        - Mexico
            - MexicoDF
            - RivieraMaya
    - CentralAmerica
        - Bahamas
            - Nassau
        - Barbados
            - Bridgetown
        - CostaRica
            - SanJosé
        - Cuba
            - Havana
            - Varadero
        - DominicanRepublic

        - SantoDomingo
        - PuertoPlata
        - PuntaCana
    - FalklandsIslands
        - Stanley_Port
    - Haiti
        - PortPrince
    - Jamaica
        - Kingston
    - PuertoRico
        - SanJuan
    - TrinidadAndTobago
        - SpainPort
- SouthAmerica
    - Argentina
        - BuenosAires
    - Bolivia
        - LaPaz
    - Brazil
        - Brasilia
        - RioDeJaneiro
        - SaoPaulo
    - Chile
        - SantiagoOfChile
    - Colombia
        - Bogotá
    - Ecuador
        - Quito
    - Paraguay
        - Asunción
    - Peru
        - Lima
    - Uruguay
        - Montevideo
    - Venezuela
        - Caracas
- Asia
    - Afghanistan
        - Kabul
    - Bahrein
        - Manama
    - China
        - Beijing
    - India
        - NewDelhi
    - Iran
        - Tehran
    - Iraq
        - Bagdad
    - Israel
        - Jerusalem
    - Japan
        - Tokyo
    - NorthKorea

        - Pyongyang
    - Pakistan
        - Islamabad
    - Philippines
        - Manila
    - Qatar
        - Doha
    - SaudiArabia
        - Riad
    - SouthKorea
        - Seoul
    - Syria
        - Damascus
    - Thailand
        - Bangkok
    - Vietnam
        - Hanoi
- Europe
    - Andorra
        - AndoraLaVella
    - Austria
        - Vienna
        - Salzburgo
    - Belgium
        - Brussels
        - Bruges
        - Lieja
    - Bulgaria
        - Sofia
    - Croatia
        - Zagreb
    - Cyprus
        - Nicosia
    - CzechRepublic
        - Prague
        - KarlovyVary
    - Finland
        - Helsinki
    - France
        - Paris
        - Bordeaux
        - Lyon
        - Marseilles
        - Toulouse
    - Germany
        - Berlin
        - Munich
        - Frankfurt
        - Hamburg
        - Dortmund
    - Greece
        - Athens
    - Holland

    - Amsterdam
    - Rotterdam
    - Eindhoven
- Hungary
    - Budapest
- Iceland
    - Reykjavik
- Italy
    - Rome
    - Venice
    - Florence
    - Milan
    - Turin
- Luxembourg
    - LuxembourgCity
- Monaco
    - MonacoCity
- Norway
    - Oslo
    - Bergen
    - Lillehammer
- Poland
    - Warsaw
    - Cracovia
- Portugal
    - Lisbon
    - Oporto
    - Coimbra
- Romania
    - Bucharest
- Russia
    - Moscow
    - SaintPetersburg
- SanMarino
    - SanMarinoCity
- SerbiaAndMontenegro
    - Belgrado
    - Podgorica
- Slovakia
    - Bratislava
- Spain
    - BalearicIslands
        - Mallorca
            - *hasAccommodation = SolBarbados*
            - *hasAccommodation = SolAntillas*
            - *hasAccommodation = MeliaDeMar*
        - Menorca
        - Ibiza
        - Formentera
        - Cabrera
    - CanaryIslands
        - Tenerife
            - *hasActivity = Bathing*

- *hasActivity = Sunbathing*

- Lanzarote
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*
  - *hasAccommodation = GranMeliaVolcanLanzarote*
- Fuerteventura
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*
  - *hasAccommodation = SolGorriones*
  - *hasAccommodation = IberostarPlayaGaviotas*
- GranCanaria
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*
- LaPalma
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*
- ElHierro
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*
- LaGomera
  - *hasActivity = Bathing*
  - *hasActivity = Sunbathing*

- Peninsula
  - Madrid
    - *hasAccommodation = MeliaAvenidaDeAmerica*
    - *hasAccommodation = NHJoseAbascal*
    - *hasAccommodation = TrypAlondras*
  - Barcelona
  - Sevila
  - Valencia
  - Málaga
  - Zaragoza
  - Coruña
  - Gibraltar

- Sweden
  - Stockholm
- Switzerland
  - Bern
- Turkey
  - Ankara
  - Istanbul
- Ukraine
  - Kiev
- UnitedKingdom
  - London
  - Cambridge
  - Oxford
  - Manchester
  - Birmingham
- VaticanCity
  - Vatican

- Oceanía
  - Australia

        - Sydney
        - Canberra
    - Fiji
        - Suva
    - NewCaledonia
        - Noumea
    - NewZealand
        - Wellington
    - SalomonIslands
        - Honiara
    - Samoa
        - Apia

- Season
    - Spring
        - March
        - April
        - May
    - Summer
        - June
        - July
        - August
    - Autumn
        - September
        - October
        - November
    - Winter
        - December
        - January
        - February

- Transport
    - AirTransport
        - Helicopter
        - Plane
        - SmallPlane
    - GroundTransport
        - Bicycle
        - Bus
        - Car
        - Caravan
        - Coach
        - Motorcycle
        - Train
    - SeaTransport
        - AquaticMoto
        - Boat
        - Cruise
        - SailBoat
        - Yacht

Propiedades

hasAccommodation

- Dominio = Destination
- Rango = Accommodation
- Inversa = isLocatedAt

hasActivity

- Dominio = Destination
- Rango = Activity

hasAccommodation

- Dominio = Accommodation
- Rango = AccommodationRating

hasAccommodation

- Dominio = Accommodation
- Rango = Destination
- Inversa = hasAccommodation